Generating Volatility Surfaces using Variational Autoencoders

Michael Barasciutti, Axel Fossum

Master's thesis
2024:E22

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

EXAMENSARBETE
Matematik

2024:E22

Generating Volatility Surfaces using Variational Autoencoders

Generering av Volatilitetsytor med hjälp av Variational Autoencoders

Michael Barasciutti, Axel Fossum

Generating Volatility Surfaces using Variational Autoencoders

(A fast way to generate volatility surfaces)

Michael Barasciutti
michaelbarasciutti@gmail.com

Axel Fossum
axel.fossum@gmail.com

May 28, 2024

Master's thesis work carried out in collaboration with Nordea Markets, Copenhagen.

Supervisors: Jimi Truelsen, jimi.truelsen@nordea.com
Alexandros Sopasakis, alexandros.sopasakis@math.lth.se
Oskar Åström, oskar.astrom@math.lth.se

Examiner: Karl Åström, karl.astrom@math.lth.se

Abstract

This study presents an in-depth exploration into the utilization of Variational Autoencoders (VAEs) for modeling and completing implied volatility surfaces within the context of the index equities market, a crucial aspect of option pricing. Moreover, our study examines the predictive capabilities of neural networks concerning fluctuations in spot prices, with a specialized spot model calibrated to forecast changes in volatility surfaces based on spot price dynamics. Through comprehensive data processing and structuring of VAEs we created a model capable of generating accurate and nearly arbitrage-free volatility surfaces from as little as 10 points of information. This model also proved proficiency in generating volatility surfaces for previously unseen underlying assets. Applying changes in spot price as a conditional variable we successfully created a powerful risk management tool capable of forecasting volatility surfaces for various future scenarios.

Although our model can be improved upon, our findings underscore the robustness and generalizability of VAEs, showcasing their potential for broader application across various financial instruments and markets.

Keywords: Variational Autoencoder (VAE), static arbitrage, implied volatility, volatility surface, index equity options, neural network, machine learning, quantitative finance

Acknowledgements

We would like to thank Alexandros Sopasakis and Oskar Åström at LTH for being excellent supervisors and for guidance regarding the creation of the Variational Autoencoder. We would also like to thank Jimi Truelsen at Nordea Markets for supplying the data, inspiring this project, and taking the time to bounce ideas with us to solve this non-trivial problem. We would also like to thank Maxime Bergeron for interesting reads in regard to VAEs in the FX markets and for answering our questions by email. Finally, we would like to thank our friends and family for their encouragement throughout the thesis.

Contents

Chapter 1
Introduction

Modeling of the implied volatility surface is an important subject for most quantitative trading firms. The volatility surface consists of the implied volatilities from inverting the Black & Scholes formula for options at different maturities and strikes. The calculations to model volatility surfaces are very complex, therefore neural networks are looked at as an alternative. Once neural networks are trained, they possess the ability to rapidly generate new surfaces. The primary challenge, however, lies in ensuring the accuracy and precision of these newly generated surfaces, particularly when constrained by the availability of limited input data. This is where Variational Autoencoders (VAEs) can be useful. Variational Autoencoders is a type of neural network that is considered a generative network, meaning that once trained, it can easily output similar patterns to what it has been trained on. Therefore by training it on historical volatility surfaces, the network should be able to output similar-looking volatility surfaces.

When modeling a volatility surface, an important consideration is the presence of arbitrage. If the surface is not arbitrage-free, competing investors will be able to obtain risk-free and cost-free profits by a combination of buying and selling different options with different expiries and strike prices. Therefore a big emphasis when creating models is to ensure that the generated surface is arbitrage-free. The volatility surfaces generated by the model are integrated into more comprehensive models that assume an absence of static arbitrage, which heightens the importance of producing an arbitrage-free surface.

Volatility surfaces play a pivotal role in understanding the dynamics of spot prices in financial markets. The relationship between volatility surfaces and spot price movements is intricate and multifaceted. Understanding how volatility surfaces interact with spot prices is something that neural networks could help to investigate.

1.1 Related Work

A lot of work has been done on producing arbitrage-free volatility surfaces. The most prominent research in this area can be found in [12] where Jim Gatheral and Antoine Jacquier introduce conditions to the SSVI model, an extension of the SVI model, to completely remove static arbitrage. This model has been widely used in the finance industry since it was published and is a basis for many research papers regarding the generation of arbitrage-free volatility surfaces such as [1, 22].

The utilization of neural networks for calculating implied volatility is becoming increasingly common in the calibration of financial models. Different types of neural networks have been applied, and different approaches have been considered for the task of volatility model calibration and implied volatility surfaces generation [2, 10, 3, 18, 7]. In [2] a neural network-based approach is used to calculate rough volatility. In this paper, a two-step approach is suggested where a neural network is utilized as the first step in approximating prices and a traditional model calibration algorithm as the second step to calculate volatility.

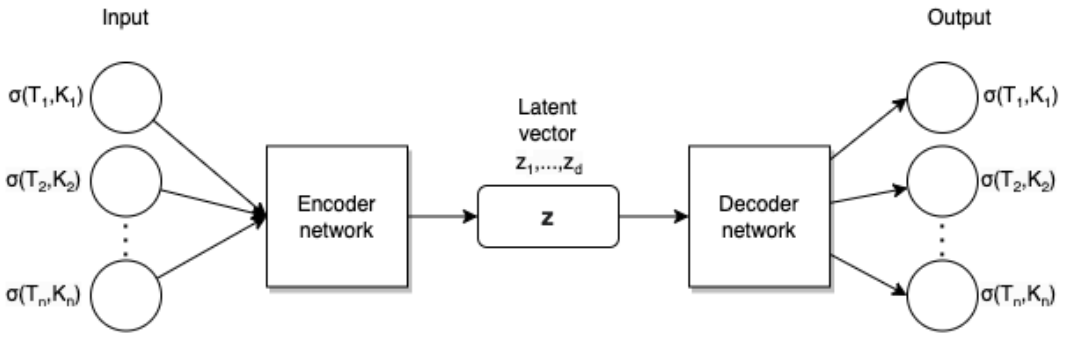
In [10] convolutional neural networks (CNNs) are used to replicate the calibration of the Heston model to equity volatility surfaces. The input to the CNN is the implied volatility combined with a 3-dimensional tensor featuring additional information, namely, the strike, the moneyness, and the equity forwards. The target for the model to reproduce is the 5 parameters of the Heston model, and the paper shows promising results in doing so.

In papers [3, 18] Variational Autoencoders (VAEs) are trained to generate arbitrage-free volatility surfaces for the FX market. In [18] a hybrid method is used combining VAEs with stochastic differential equation models (SDE models). In the paper, a VAE is trained using SDE model parameters from previously known surfaces. To generate IV surfaces a sample is drawn from the latent space of the VAE which is then decoded to generate SDE model parameters, which are then mapped to IV surfaces. They then further improve the VAE by turning it into a conditional variational autoencoder by introducing conditional features, as first introduced in [21].

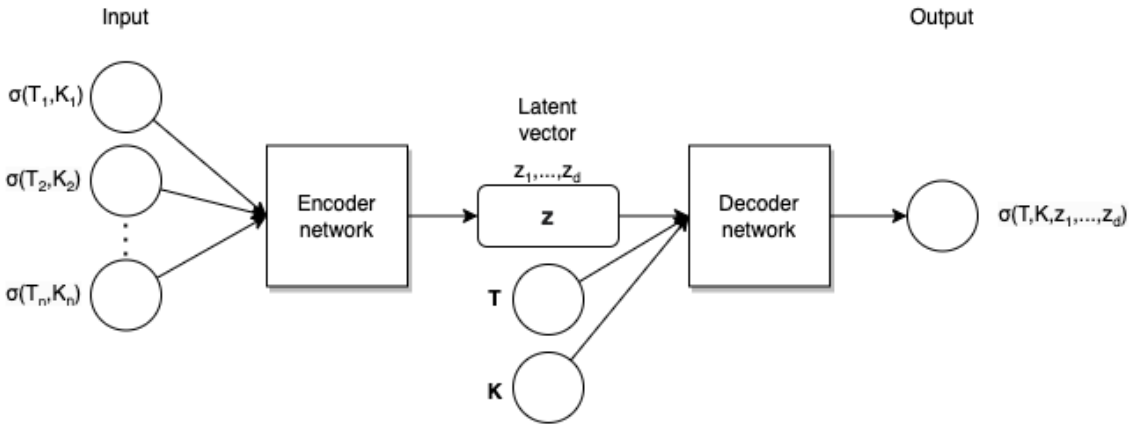
In Bergeron et al.[3] the input to the VAE consists of a grid of 40 volatility points. It consists of volatility for deltas (0.1, 0.25, 0.5, 0.75, and 0.9) for the expiries (one week, one month, two months, three months, six months, nine months, one year, and three years). The network is then trained to reconstruct points on the grid using different sizes of the latent space. The VAE trained can with the help of an external optimizer create reasonable-looking volatility surfaces being fed as little as 5 points for the FX market.

In [3] inspired by suggestions in [2], two different approaches for the VAE are presented; a gridwise, and a pointwise approach. The gridwise approach, as illustrated in Figure 1.1a is a typical VAE that takes a volatility surface with specified maturities and deltas as input and outputs a volatility surface. However, this lacks the capability of inferring points with different delta and tenor values and instead, the pointwise approach is suggested. As illustrated in Figure 1.1b the delta (K) and maturity (T) are added as extra information to the latent vector fed to the decoder to have the network output a single point. This now turns the network from an unsupervised network into a semi-supervised network and enables training and inference on points outside the specified points inputted in the grid. Whereas the gridwise approach relies on an outside interpolation to compute the implied volatility of arbitrary deltas and maturities, as discussed in [2].

There also exists previous work where neural networks have been used to predict volatility surface movements based on changes in spot price combined with the underlying volatility as seen in [7]. Through using 3 hidden layers with 80 nodes each they are able to accurately predict volatility surfaces and their research strengthens previous theory stating a negative correlation between volatility levels in the SPX and changes in spot price. They do however notice exceptions to this theory. While under normal circumstances a positive return in the underlying asset would lead to a decrease in volatility for the entire volatility surface, the opposite seems to be true if the spot movements happen during an already low volatility regime. This is particularly noticeable in the high-delta short-maturity options. These findings would imply that the entire shape of the volatility surface and not only the general volatility level is highly impacted by changes in the underlying spot price.



(a) Gridwise approach



(b) Pointwise approach

Figure 1.1: An illustration of the architecture of two Variational Autoencoders one implementing the gridwise, and one implementing the pointwise approach.

1.2 Research Objective

In this master's thesis, we aim to explore the feasibility of a Variational Autoencoder (VAE) capable of effectively constructing arbitrage-free volatility surfaces within the equities market, by utilizing historical SPX option data as training data. We also want to investigate the applicability and performance of such a model on indices not included in the training data. Furthermore, we aim to explore the latent space of such a model to acquire knowledge of how the VAE represents these surfaces. Another goal is to examine the potential for developing a model that can predict alterations to the volatility surface in response to movements in the spot price. We place a big emphasis on our models being arbitrage-free since it is important for practical use.

1.2.1 Contribution to Research

Numerous papers explore the application of neural networks for generating volatility surfaces, as evidenced by works such as [2, 10, 22]. Notably, VAEs have been employed to model volatility surfaces within the FX market context, as detailed in studies [3, 18]. With this thesis, we aim to investigate the extension of these neural networks, specifically VAEs on the equities market which represents a more difficult problem due to the difference in price quoting and liquidity of the market. We want to further extend the applicability of these models to accurately predict changes in the volatility surfaces in response to price movements in the underlying assets similar to [7].

1.2.2 Contribution Statement

During the creation of the model and data processing, work was carried out in a pair-programming fashion. This ensures that both of us have complete knowledge of how our solution works. This has been a good way to catch mistakes at an early stage and has facilitated discussion on how problems should be solved. The writing of the thesis has been executed in a similar fashion with some parts being completely written together and some parts initially written by one person and rewritten by the other.

Chapter 2

Theory

This chapter features all the background theory used to understand the project. It includes both theory on options, arbitrage, and variational autoencoders.

2.1 Options Theory

2.1.1 Implied Volatility

Options are sophisticated financial derivatives that give investors the opportunity to speculate and bet on price movements among other factors in the underlying asset. This study focuses on European index options. A European option gives a buyer the right to sell or buy the underlying asset for a set price at a set time. European options are the most commonly traded option type in the index space and contrary to American options, the European counterpart can not be exercised prior to its maturity date. An index is a statistical measure that represents the performance of a collection of assets. The index is constructed by aggregating the returns of these assets, and each asset is weighed differently depending on which method the index chooses to use. In this project, the focus is primarily on equity indices, especially the S&P 500 which represents the largest 500 stocks in the U.S. and is the most traded equity index in the world. The S&P 500 represents the aggregated returns of these 500 stocks weighed by their total market capitalization [6].

Crucial to options trading is the pricing of these options at any time and moment. The pricing model most widely used is the Black and Scholes model [5] as seen in Equation (2.1).

$$C(S_t, t) = S_t N(d_1) - K e^{-r(T-t)} N(d_2), \tag{2.1}$$

where:

S_t is the price of the underlying asset at time t

$C(S_t, t)$ is the call option price at time t

$N(x)$ is the cumulative distribution function of the standard normal distribution

$$d_1 = \frac{\ln(S_t/K) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

K is the strike price

r is the risk-free interest rate

T is the time to expiration

t is the current time

σ is the volatility of the underlying asset

This model takes into account a variety of factors including the spot price of the underlying asset, the interest rate for the maturity period, the options strike price, the time remaining until expiry, and the volatility of the underlying asset. The parameter that considers the volatility of the underlying asset is called the implied volatility and can be seen as a forecast of the future volatility during the options life span. In the Black-Scholes framework, it is assumed that stock prices follow a geometric Brownian motion, which means that the logarithm of stock prices follows a normal distribution over time. In reality, stock returns often exhibit characteristics such as skewness, kurtosis, and volatility clustering that deviate from a perfectly normal distribution [14]. This key assumption is the biggest flaw in the model and is the reason why accurate option pricing is such a difficult task. To accommodate for real-life assumptions about the options' probability of exercise, speculators have to incorporate modifications or rely on alternative models for an accurate price. One modification is to change the implied volatility parameter of each option depending on its strike price and maturity date. Since all other parameters are more or less known, selecting the correct implied volatility becomes the biggest challenge in option pricing, and most professional trading is done based on different opinions on the correct implied volatility.

2.1.2 Incorporating Dividends

The Black & Scholes formula is not built to incorporate dividends so alterations are required before using it to price index options. Since this project is concerned with equity indices consisting of large numbers of underlying stocks we will consider the dividend rate near continuous. To accommodate for dividends, the forward price of the underlying asset is calculated by discounting the current spot price with the continuous dividends [14] as in Equation (2.2).

$$F = S_0 \cdot e^{r-d \cdot T}, \tag{2.2}$$

where:

- F is the forward price.

- S_0 is the current spot price of the asset.
- d is the continuous dividend yield (dividend payment rate).
- r is the continuous risk free interest rate .
- T is the time to expiration of the forward contract.

2.1.3 Risk Parameters of Options

Another important aspect of options is the relative ease with which investors can calculate their risks by using the partial differential equation from the Black & Scholes Formula. In this study, two specific risk measurements namely delta and vega will be referred to. Delta is the measurement of how the option price will move based on price movements in the underlying asset [14] and the formula can be seen in Equation (2.3). This paper will mostly refer to options with specific deltas instead of specific strikes from this point forward as the delta values are already normalized which allows for easier use as input to a variational autoencoder. Modeling based on delta also allows us to consider the full volatility surface which concerns investors. If the volatility surfaces were modeled off fixed strike prices it would be difficult to capture the high and low delta-valued options at higher tenors when a rectangular grid is desired.

The delta of a European call option is given by:

$$\Delta = N(d_1), \tag{2.3}$$

where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}. \tag{2.4}$$

The vega of an option is the first derivative of the Black & Scholes formula with regards to the volatility parameter and represents the option's price movement based on the change in volatility in the underlying asset[14]. Vega will be used to invert the Black & Scholes formula to obtain implied volatilities. The formula for vega can be found in Equation (2.5).

The vega of a European option is given by:

$$\text{vega} = S \cdot \sqrt{T} \cdot N(d_1). \tag{2.5}$$

2.1.4 Volatility Surfaces

Since the implied volatility of an option depends on its strike price and maturity date, a volatility surface is created by visualizing all possible option prices. The very existence of a non-flat volatility surface proves that the Black and Scholes formula is far from perfect. While looking at historical market data, it can often be observed that market prices fall rapidly but rise slowly. Due to this, the probability distribution of where the underlying price may end on the expiration date can not be normally distributed. The exact shape of the surface is determined by market sentiment at the time. During periods of turmoil, for example, during the beginning of the covid crisis, investors were constantly on guard for incoming negative

news which might cause the market to fall rapidly. This led to a volatility smirk as seen in Figure 2.1 where investors assigned a higher probability for the underlying asset to finish with a spot price closer to the lower strikes [4].

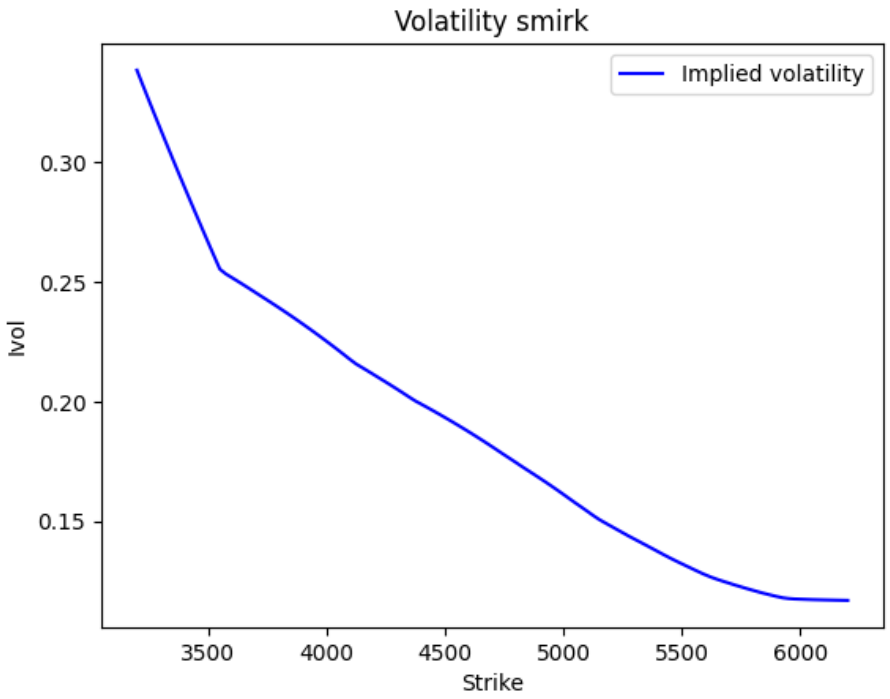
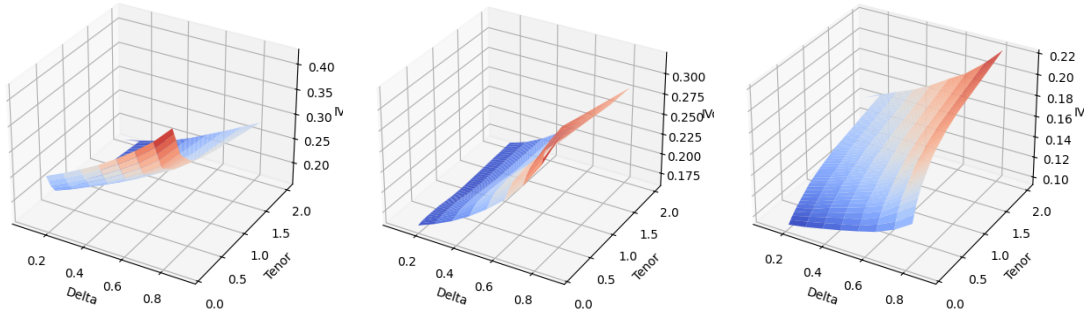


Figure 2.1: Volatility smirk of one specific tenor. Lower strikes are assigned a higher probability and therefore the implied volatility will be priced higher.

The lifetime of the option also has to be considered when pricing the implied volatility. Suppose the market is in a high volatility period at the moment but investors are expecting future volatility to be lower. In that case, the options with shorter tenors will have to be assigned higher implied volatility than the options with longer tenors. The opposite is also true and more commonly seen in the equity index market. Volatility may be low today, however, the volatility of the future might increase due to uncertainties and speculators will then have to price longer tenor options with higher implied volatilities. In Figure 2.2 volatility surfaces can be seen that showcase how different the volatility surface of options on the same underlying asset can look depending on market sentiment.



(a) **(b)** **(c)**

Figure 2.2: This figure illustrates three different types of volatility surfaces on SPX options. Figure a) displays a common volatility surface during periods of high volatility such as during the covid crisis. Figure b) displays a volatility surface common during transition periods between different volatility regimes. Figure c) displays the most commonly found volatility surface for periods of usual volatility levels.

2.2 Modelling of Volatility Surfaces

How each speculator chooses to price their implied volatility surface is different depending on their own proprietary research, however some baseline models are widely used. The stochastic volatility-inspired parametrization of the implied volatility smile is one of the most popular ways to price volatility since it is a rather simple model and can easily be kept free of calendar spread arbitrage. As seen in [12] the SVI parametrization can also be extended to be free of butterfly arbitrage. By having the SVI model free of both calendar spread and butterfly arbitrage, it can also be proved that the model is completely free of static arbitrage as seen in [12]. The SVI parametrization works by parameterizing the implied volatility for one specific tenor, and once you have the parameters optimized, it is easy to price implied volatility for all strikes. An issue with the raw SVI model is that the parameters need to be fitted for each tenor separately which also gives cause to no natural way of pricing options outside the pre-fitted tenors. Fitting the parameters for each tenor individually can be costly and computationally heavy as well. Due to there being 5 parameters per tenor, the model will require a significant amount of parameters to cover the entire surface which might not be ideal due to overfitting and computational reasons.

The formula for the raw SVI parametrization of the implied volatility smile is given by:

$$w(k;\chi_R) = a + b\left(\rho \cdot (k-m) + \sqrt{(k-m)^2 + \sigma^2}\right), \tag{2.6}$$

where:

$w(k;\chi_R)$ is the implied volatility for strike k and parameters χ_R
a, b, ρ, m, σ are parameters to be calibrated to market data

While the raw SVI parametrization is usually not extended to fit an entire surface, the natural SVI parametrization can be. The natural SVI parametrization is the functional form

that appears from the limit of the Heston model as seen in [11]. This model is however not as useful as the raw parametrization due to the unaesthetic representations of ω and η from the Heston model as seen in Equation (2.7).

The natural SVI parametrization is given by:

$$w(k;\chi_N) = \Delta + \frac{\omega}{2} * (1 + \zeta\rho(k-\mu) + \sqrt{(\zeta(k-\mu)+\rho)^2 + (1-\rho^2)}, \tag{2.7}$$

where Δ, ω, ρ, ζ, q, and μ are parameters.

2.3 Arbitrage

Arbitrage refers to a cost-free trading strategy resulting in a positive probability of obtaining risk-free profitable results. Arbitrage occurs due to discrepancies and inaccuracies in pricing mechanisms between markets. In this project scope, only static arbitrage is considered. Unlike dynamic arbitrage where investors have to continuously adjust positions to keep arbitrage, static arbitrage refers to taking advantage of price differences without the need for ongoing adjustments. This is especially relevant in option pricing where inaccuracies in volatility surfaces or between put and call prices can lead to static arbitrage opportunities. Static arbitrage is rather rare and very short-lived since investors will quickly exploit these inaccuracies and prices will revert to non-arbitrage levels. The cornerstone for arbitrage is the put-call parity which specifies the relation between the put price, the call price and the underlying asset as seen in Equation (2.8). Assuming a liquid tradeable underlying asset the put-call parity has to hold for there to be no tradeable combination of the put, call, and underlying resulting in arbitrage. During conditions where the underlying is not tradeable or trades at a large spread, many arbitrage opportunities become unfeasible. In this paper, a tradeable underlying asset with high liquidity is assumed.

Put-Call Parity with Continuous Dividends and Interest Rates:

$$C - P = S_0 e^{-qT} - Ke^{-rT}, \tag{2.8}$$

where:

C is the price of the call option,
P is the price of the put option,
S_0 is the current price of the underlying asset,
K is the strike price of the options,
q is the continuous dividend yield,
r is the continuous risk-free interest rate,
T is the time to expiration.

As can be seen in [12] a volatility surface is only free of arbitrage if and only if the following conditions are met.

(i) It is free of calendar spread arbitrage.

(ii) Each time slice is free of butterfly arbitrage.

Specifically, the absence of butterfly arbitrage guarantees the presence of a probability density that is non-negative, while the absence of calendar spread arbitrage indicates that option prices exhibit monotonicity concerning maturity.

Calendar spread arbitrage involves purchasing a call option C_2 with a longer maturity date and simultaneously selling a call option C_1 with a shorter maturity date, where the call prices are denoted as $c_1, c_2, \ldots, c_n$ with assigned strike prices $k, k_1 < k_2 < \ldots < k_n$. Mathematically, a calendar spread is defined as $CS = C_2 - C_1$.

Butterfly spread arbitrage involves simultaneously buying two options and selling one option, all with the same expiration date, but with three different strike prices. Specifically, it entails buying one call option C_1 with a lower strike price, buying one call option C_3 with a higher strike price, and selling two call options C_2 with a strike price in between C_1 and C_3. The call prices are denoted as c_1, c_2, c_3, and the assigned strike prices are $k_1 < k_2 < k_3$. Mathematically, a butterfly spread is defined as $BS = C_1 + C_3 - 2C_2$.

As can be read in [8] a rectangular grid of European call prices can be said to be free of all static arbitrage if all adjacent call spreads and butterfly spreads are non-negatively priced. Since the calculations for detecting static arbitrage in the implied volatility space are quite long and very well documented already we refer to [12] [9] for more in-depth explanations, however, it will be sufficient to convert the implied volatilities into call prices and check for arbitrage for the scope of our project.

2.4 Arbitrage Repair

Due to the frequent presence of arbitrage when considering mid-prices of options, implementing an arbitrage repair algorithm may be necessary for accurate modeling. A major difference between arbitrage repairing and arbitrage smoothing is that while arbitrage smoothing generally changes most of the data points, arbitrage repairing aims to change the points where the issue lies. The major challenges in repairing arbitrage are the number of constraints required as well as making sure the algorithm does not alter the data outside reasonable bounds. Due to this, we chose to utilize an algorithm that incorporates the bid and ask spreads as soft bounds with the arbitrage constraints as hard bounds as seen in [9]. Before any calculations, all variables are normalized. The use of different norms for measuring perturbations in option prices is the first thing that needs to be decided upon. While the l_2 norm is most commonly used in cases of price repairing due to its convexity and computational efficiency it may not always produce sparse solutions. The l_0 norm usually leads to sparse solutions but has the downside of NP-hard optimization problems. Therefore the l_1 norm, the convex relaxation of the l_0 norm is chosen as an alternative due to its robustness to outliers and ability to produce sparse solutions. The objective function proposed in [9] aims to minimize perturbations in option prices while ensuring they remain within bid-ask price bounds. This objective function is expressed as a sum of individual cost functions for each option price, where the cost function accounts for the bid-ask price spread and retains the

ability to be able to be written as an LP problem. The specific function can be seen below.

$$f(\epsilon) = \sum_{j=1}^{N} \max\left(-e_j^T\epsilon - \delta_j^b + \delta_0, -\frac{\delta_0}{\delta_j^b}e_j^T\epsilon, \frac{\delta_0}{\delta_j^a}e_j^T\epsilon, e_j^T\epsilon - \delta_j^a + \delta_0\right), \tag{2.9}$$

where N is the number of options, ϵ_j is the perturbation of the j-th option price, δ_j^a and δ_j^b are the bid and ask reference spreads for the j-th price, respectively, and δ_0 is a constant representing the bid-ask spread and is chosen as $\delta_0 = \frac{1}{N} \wedge \min_{j=1,\dots,N}(\delta_j^a \wedge \delta_j^b)$. e_j is the standard basis vector for R^N with its j-th element being 1 and others being 0. In practice, this means that we would rather move every option price by ϵ rather than moving one single option outside the bid-ask spread.

The repair problem can now be defined as the following LP by introducing auxiliary variables $t = [t_1, \dots, t_N]^T$:

$$\text{minimize} \quad \epsilon, t \quad \sum_{j=1}^{N} t_j, \tag{2.10}$$

subject to:

$$\begin{aligned}
&-\epsilon_j - \delta_{bj} + \delta_0 \le t_j, && \epsilon_j - \delta_{aj} + \delta_0 \le t_j, && \forall j \in [1, N], \\
&-\frac{\delta_0}{\delta_{bj}}\epsilon_j \le t_j, && \frac{\delta_0}{\delta_{aj}}\epsilon_j \le t_j, && \forall j \in [1, N], \\
&-A\epsilon \le -b + Ac. &&&&
\end{aligned} \tag{2.11}$$

The normalized arbitrage-free call prices can now be calculated by adding ϵ to the original normalized call prices. The call prices are then de-normalized and used for further calculations. For further details and proofs, we refer to [9].

2.5 Variational Autoencoders

A variational autoencoder is a type of neural network based on an autoencoder. The structure of a typical autoencoder is illustrated in Figure 2.3. The size of the output layer of an autoencoder should equal the size of the input layer. That is a very trivial problem to solve using a neural network with a hidden layer the same size as the input. In autoencoders, the hidden layer is therefore required to be of a smaller size than the input layer to ensure that dimensionality reduction is done. The first part of the network that does the dimensionality reduction is called the encoder and the part that reconstructs the output from the encoder is called the decoder as seen in Figure 2.3. The intermediate layer between the encoder and decoder is known as the latent encoding. The encoder's objective is to generate a latent encoding that enables the decoder to reconstruct the input data with minimal loss.[19]

What differentiates the variational autoencoder from an autoencoder is what steps are performed in this latent encoding. In the VAE as introduced by Kingma and Welling [15] explicit uncertainty is introduced to the creation of the latent encoding through a reparameterization step. From the encoding layer, the output is now two vectors a mean vector μ, and a standard deviation vector σ as seen in 2.4. In the reparameterization step to create the latent vector z, sampling is done from the distribution created by μ and σ vectors. However,

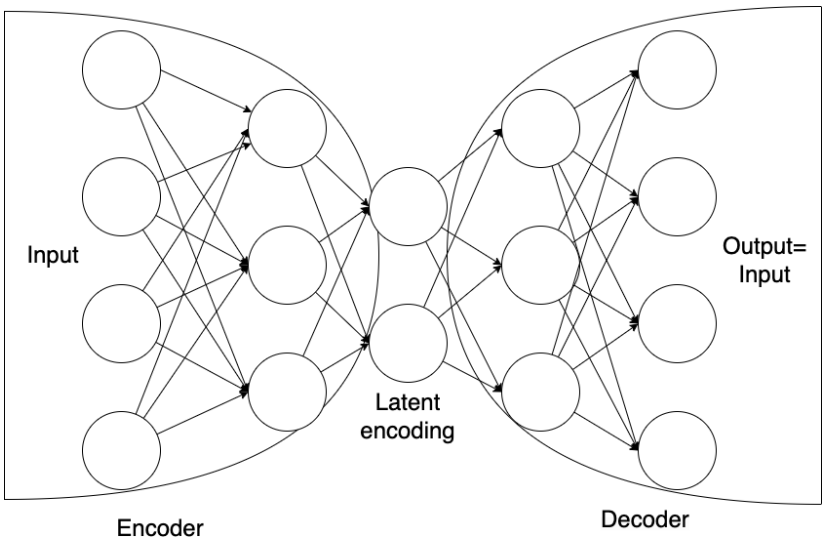


Figure 2.3: An illustration of the autoencoder structure highlighting the encoder, latent encoding, and decoder.

to be able to use the backpropagation algorithm on μ and σ a variable ϵ is introduced, which is a sample from the unit normal distribution $\mathcal{N}(0, 1)$. We get

$$z_n = \mu_n + \sigma_n \epsilon \tag{2.12}$$

Where σ_n and μ_n are the standard deviation and mean for the nth latent variable z_n.

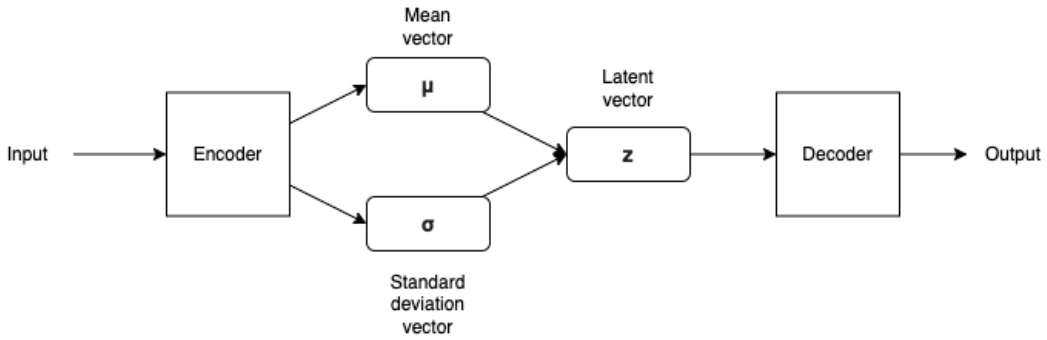


Figure 2.4: An illustration of the VAE structure featuring the standard deviation vector σ and mean vector μ.

A prior distribution for the latent variables is used. A commonly used prior distribution is a multivariate normal distribution, denoted as $\mathcal{N}(0, 1)$, where the variables are uncorrelated, have a mean of zero, and a standard deviation of one. If the loss of the autoencoder is calculated by only comparing the input and output σ would likely converge to zero for all inputs during training since adding any variation the optimal μ would be worse. Therefore an additional term is added to the loss function, the Kullback-Leibler divergence (KLD) between $\mathcal{N}(\mu_n, \sigma_n)$ and $\mathcal{N}(0, 1)$ to penalize disagreement between the two distributions and train the autoencoder closer to the unit normal distribution. The KLD term is defined as

$$L_{KLD} = \frac{1}{2}\sum_{n=1}^{N}\left(1 + \log(\sigma_n^2) - \mu_n^2 - \sigma_n^2\right), \tag{2.13}$$

where σ_n and μ_n are the standard deviation and mean for the nth latent variable z_n.
The complete loss function therefore is

$$L_{RE} + \beta L_{KLD}, \tag{2.14}$$

where L_{RE} is the reconstruction error, the mean squared error (MSE), and β is a hyperparameter used to tune the importance of the KLD loss, L_{KLD}. Adding the KLD term and training towards a unit normal distribution enables the generation of new reasonable samples by feeding the decoder a different latent vector z.

Activation Functions

A popular choice of a non-linear activation function for deep neural networks and VAEs is the rectified linear unit (ReLu). It provides fast calculations and deals with vanishing gradients well. [19]. The ReLu function is shown below

$$\varphi(x) = \max(0, x), \tag{2.15}$$

where x is the input to the activation function.
An extension of the ReLu activation function is the Leaky ReLu, which features a small slope for negative values instead of setting them to zero. It has grown popular in many applications, not the least in training general adversarial networks (GAN) as in [20]. The formula for the Leaky ReLu is shown below

$$\varphi(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise,} \end{cases} \tag{2.16}$$

where α is a coefficient set before training and x is the input to the activation function as first introduced in [17]

Optimizers

Minimizing the loss function is done by the backpropagation algorithm using optimizers. A popular choice for an optimizer is Adam which allows for training in batches. Adam has three different hyperparameters for learning η (learning rate), β_1, and β_2. Tuning these parameters in addition to batch size is a good way to avoid getting stuck in local minima when minimizing the loss function.[19]

Overfitting and Regularization

A problem often faced when training neural networks is overfitting. The problem originates from the network being trained to perform on the training data and as a result, performs worse on new input outside the training data. Overfitting can become a problem especially when featuring many different hyperparameters that are tuned to optimize performance on

the training set. To combat this, several different strategies are used to reduce the loss on the test set, called regularization, potentially at the expense of higher training loss. [13] A common regularization method is L2 regularization.

L2 regularization combats overfitting by modifying the loss function by punishing large weights using the L2-norm. The new loss function featuring weight decay now becomes

$$\tilde{E}(\omega) = E(\omega) + \alpha\Omega(\omega) \tag{2.17}$$

$$\Omega(\omega) = \frac{1}{2}\|\mathbf{w}\|_2^2, \tag{2.18}$$

where α is a hyperparameter to weigh the importance of the weight decay. [19]

L2 regularization and weight decay can be used in tandem with the Adam optimizer. One algorithm introduced in [16] decouples the weight decay from the optimization steps taken w.r.t. the loss function. This is shown to improve Adams's generalization performance significantly. This optimizer is referred to as AdamW and in addition to previous hyperparameters, Adam also features a parameter for weight decay.

Chapter 3
Method

3.1 Data

Our dataset consists of SPX (S&P500) option quotes sampled once per day during the period 2013-2023. This will be the dataset on which we train our model and validate the results. We also have similar datasets for options on the indices DAX, NDX, UKX, SX5E which will be used to test the model's ability to adapt to surfaces previously not seen. Additionally, we have fixed interest rates and spot prices for all underlying indices during the same time period. The consistency in which maturity dates and strikes are quoted for the SPX in our dataset is varied. Some years quote options with maturity dates as short as a few days while other parts of the dataset will only quote options with maturity dates longer than a month. The same is true for longer maturity dates. In the dataset, the amount of options quoted from day to day also varies. This might be due to a lack of liquidity, however, some days will contain thousands of options while others merely tens of options. Due to this, we have to filter the data heavily before introducing it to our models.

3.2 Filtering of Data and Removing Static Arbitrage

The filtering of the data is done based on discoveries throughout the entire method. Firstly, we remove the tenors not present in every year which was every tenor outside the one month-two years span. Here we left some options outside the space for interpolation purposes for the years with a wider array of tenors. We chose to base our implied volatilities on the mid prices initially, however, that led to issues with static arbitrage when generating surfaces. As can be seen in [9] the mid prices of index options frequently contain static arbitrage which we do not wish our model to learn. Due to this, we decided to utilize the arbitrage repairing function seen in Equation (2.10) on the initial data. To allow this algorithm to converge we

had to reduce the amount of options for each day. In [9] they use 500 options per day and we had as many as 2000 options for certain days. This creates a problem since the complexity of the algorithm increases exponentially with the amount of options. Due to this, we removed all options that were not going to be used in the creation of the input grid. We then altered our mid prices according to the static arbitrage conditions and removed days where the data still resulted in static arbitrage. Our theory in doing so was that the VAE would only learn arbitrage-free surfaces since the training data did not contain any static arbitrage. Removing days containing arbitrage also automatically filters out days containing corrupt data and results in a cleaner dataset.

3.3 Extracting Volatilities

Our next step is to extract the implied volatility which would be used to create the training set for our model. Firstly we match each option with the risk-free rate associated with the period until maturity. Since we are working with equities we need to consider the dividends before extracting the implied volatilities. We use the formula for put call parity seen in Equation (2.8) to extract the continuous dividends for each option. Since the at-the-money options are usually the most liquid, we assumed them to have a mid-price rather close to the fair value of the option after arbitrage repairing and chose to base our dividend values for each tenor on them. We then extended the dividend values for all options in the specific tenor and calculated the forward spot price as seen in Equation (2.2). We then calculate the vega of each option as seen in Equation (2.5) to be used when numerically solving for the implied volatility. To solve for the implied volatilities we invert the Black & Scholes formula seen in Equation (2.1) and utilize Newton's method seen in B.2 to numerically solve for the values. The vegas of the options are used as the derivative in Newton's Method. Since this is a numerical solution it does not always converge to a realistic value. This often happens when the bid-ask spread is large leading to unrealistic mid prices. Therefore we remove data points with unrealistic implied volatilities or where the solution did not converge.

3.4 Creating the Grid

Due to the structure of our model, we chose to have a consistent volatility surface for each day. This volatility surface will be referred to as our input grid and consists of options at 7 fixed tenors and 5 fixed delta values. The chosen tenors were one month, two months, three months, six months, nine months, 1 year, and 2 years. The chosen deltas were 0.1, 0.25, 0.5, 0.75, 0.9. Since there were not always options in the market matching these tenors and deltas we had to refer to interpolation. Firstly we find the options closest to the desired tenors. If they are not sufficiently close we interpolate all the options between the two closest tenors using total variance. It is shown in [12] that any monotonic interpolation between two volatility smiles already free of static arbitrage will also be free of arbitrage. After obtaining a volatility smile of our desired tenor we calculate the delta values in our volatility smile according to eq. 2.3 and do the same procedure as with the tenors to acquire our desired delta value. The final result is a grid consisting of 35 arbitrage-free implied volatilities at our fixed tenor and delta values. The interpolation done in this process is restricted to using delta

values at a maximum distance of 0.03 from the desired value. If the delta value of a found option is within 0.01 of the desired value we chose not to interpolate. There are no such restrictions when interpolating between tenors. If no acceptable interpolations are found for a single point in the grid the entire day of data is discarded, although those days were infrequent.

3.5 Creating the VAE

Before creating the VAE we first did an 80/20 split on our data. We split the days in the dataset randomly. We ensured that all different types of volatility shapes were present in the training and validation set. For the pointwise approach which uses additional parameters, we normalized all tenors in the dataset. For each day we included all the points for our five deltas and seven tenors in a grid as a surface.

We had two different approaches. Starting with the gridwise approach we simply fed the surface to the VAE expecting it to reproduce the same output.

In the pointwise approach, we trained the VAE on each point separately by adding their tenor and delta values as extra parameters to the decoder and feeding every point's associated grid to the encoder. To speed up training in the pointwise approach we had every point associated with its grid allowing for much faster training in batches.

In both approaches, we used the loss function as seen in Equation (2.14), which features a term for MSE loss and a term for KLD loss. For both approaches, we use the reparametrization step seen in Equation (2.12) to generate the latent vector z. We implemented the Leaky ReLU as an activation function, anticipating that it would address the vanishing gradient problem in scenarios with a large number of layers. We used the AdamW optimizer (see B.1) that features regularization to combat overtraining, hoping that it would result in our model generalizing better.

This gave us many different hyperparameters to optimize; epochs, number of layers, layer size, dimension of the latent space, learning rate, weight decay, beta to weight the KLD loss, and batch size.

3.6 Optimizing and Validating the Model

To validate the model we mainly used two different approaches, the first approach was simply feeding the training points of the validation set to the encoder and having the decoder reproduce the surface and then calculate the MSE. The other approach was closer to how the model might be used. In this approach, we utilized a range of 10 to 35 points from the validation set and employed an optimizer, specifically employing the Nelder-Mead algorithm (refer to B.3), to determine the latent vector resulting in the minimum Mean Squared Error (MSE) for those points within the grid. We then took the entire surface produced and calculated the MSE as a validation score. The latter way of validating our model we believed to be superior since this could test how good our model was at generating surfaces from knowing only a few options of the day. We also devised a method weighing both ways of validating the model as a third method of validating the model.

To optimize our parameters we used a library called Optuna[1], which is an open-source library to optimize hyperparameter search. For both the gridwise and the pointwise approach we let this search run for eight hours to find good parameters. As validation, we used our first approach of feeding our validation patterns to the encoder and calculating the MSE from the decoder, since this approach was less computationally expensive than the second one featuring the optimizer, allowing us to test more hyperparameters. We believed this validation method to still be a good indicator of how the network would perform on other tests.

3.7 Testing the Model

As a first test to compare our models (gridwise and pointwise), we calculate the MSE using our optimizer method as mentioned above for validation with 10, 20, and 35 points on our validation set. We compare this result to that of the SVI model as a baseline. When giving the models fewer points we made sure that all the tenors had at least one point, to ensure that the SVI model would work. The exact points tested can be seen in A.1.

After obtaining a model with a good validation score we wanted to test our model in two specific qualities. We wanted to test how much arbitrage, if any, our model generated for our validation set and its ability to generalize. If our model contained a lot of arbitrage we wanted to test if adding arbitrage reparing to the output would result in a better output. Additionally, we intended to test the model's ability to generalize by producing volatility surfaces for different indices (SX5E, DAX, NDX, UKX) after training only on the S&P500.

3.7.1 Testing for Arbitrage

To be able to test for arbitrage we had to create a grid of options large enough that arbitrage could be found. The initial grid of 35 points was easily tested to be arbitrage-free, but that does not mean that all points generated inside the grid would be arbitrage-free. We decided to create a grid with the same tenors as in our input grid but implied volatilities for every 25 strikes. The lowest strike would be when the delta of the option was equal to 0.9 and the highest would be when the delta was equal to 0.1. This allows us to test for arbitrage in a grid consisting of 7 tenors and a maximum of 128 delta values. The final grid would consist of approximately 700 call prices depending on the year and market conditions. When testing for arbitrage we require the grid in a format of strike price and tenor. Since our model produces implied volatilities based on delta values and tenors we would need to find a numerical solution to produce the desired grid. We start by numerically solving the strike prices for the lowest and highest delta values in our grid using the brentq method as seen in B.4 for each day and each tenor. We then generate a list of strike prices between these two values that we would like to generate implied volatilities for. To numerically generate option prices matching our desired strikes we would first have to generate implied volatilities for a starting delta, solve for the strike price of the generated option, and then compare it to our desired strike. Our objective function can then be stated as below.

$$L = \left| K_g - K_d \right|,$$

[1]Link to Optuna is found here https://optuna.org/

where:

K_g is the generated strike price,
K_d is the desired strike price.

We then use the brentq method again to numerically solve for a delta value matching the strike price and minimize our objective function. Since the delta value depends on the generated implied volatilities and the implied volatility depends on the delta value, this becomes a rather computationally heavy process. After generating implied volatilities for all strike prices and tenors in the grid we convert them to call prices using the Black & Scholes formula. We now have everything required to test for arbitrage as seen in Section 2.3. We also test arbitrage for other cases, such as different strike intervals while restricting our model's access to the number of points on the input grid. To make sure that no arbitrage exists, we also create an algorithm where we generate the same call prices as earlier but execute the arbitrage-repair algorithm on them before testing for arbitrage. As a part of this, we also investigate how much the arbitrage-repair algorithm changed the generated volatility surfaces.

3.7.2 Testing with Other Indicies

For the other indices, we did similar filtering procedures as for S&P500 to extract volatilities and create grids. One important difference is that we did not run the algorithm to remove static arbitrage or perform any interpolation for these indices. Instead, we simply solved for the tenors and delta values closest to the values in our input grid. This allows us to have delta and tenor values in between the grid points, and we can evaluate how good our model is at generating implied volatilities for points not specifically on the grid.

For all the indices we used the year 2023 for testing since it is the most recent and relevant year. After obtaining the grid for every day, we calculated our MSE with the second validation approach using 10, 20, and 35 points. Here we also used the SVI model as a baseline to be able to compare and see if any of the indices were particularly difficult to reproduce.

3.8 The Spot Model

For the spot model, we used the same architecture as in the pointwise VAE seen in Figure 1.1b with one key distinction. In addition to adding the strike and maturity, we now added the change in spot price between the days as a parameter to the decoder. Instead of training like an autoencoder, we now used day t as input and day $t + 1$ as the output to reproduce,/ illustrated in Figure 3.1. We set the β to 0 for the spot model since we figured that any variation from the optimal μ vector would result in a worse score. We tested the spot model by feeding the encoder every day in the validation set and calculating the resulting MSE loss for the following day. When testing for arbitrage we did it in a similar fashion as for the VAE.

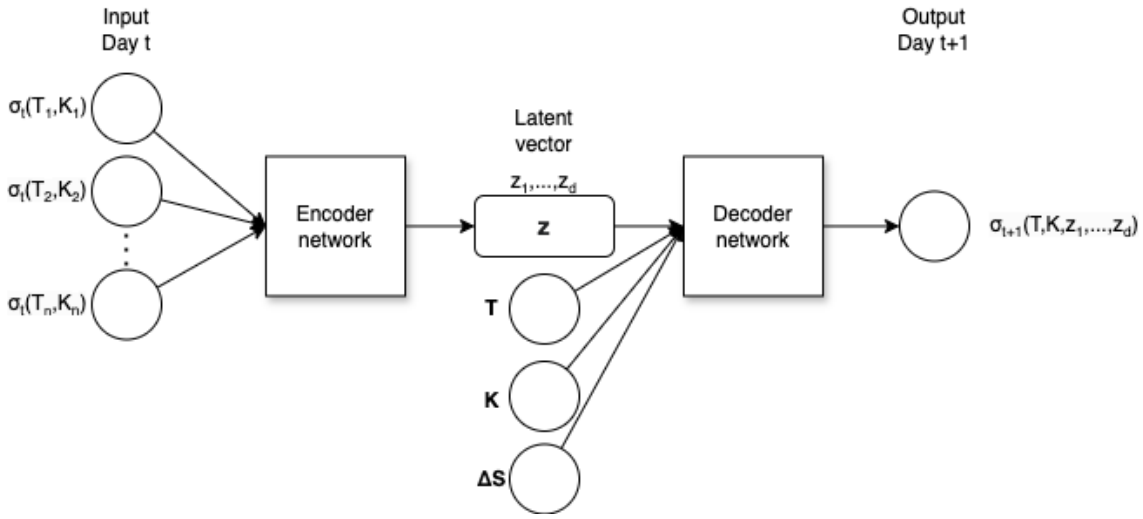


Figure 3.1: An illustration of the architecture of the spot model using day t as input and day $t + 1$ as output.

Chapter 4

Results

This chapter shows the results obtained for two different models: one trying to create a volatlity surface using a limited amount of points for a day, and one trying to create a volatility surface for the following day using a change in spot price. In addition to numerical results we also present plots of generated surfaces. We also show plots for the latent space visualization for the pointwise VAE model.

4.1 VAE Results

The following tables show the MSE losses from generating volatility surfaces on the validation and test sets. These surfaces were generated with a different number of available points while the MSE loss was calculated on the entire grid of 35 points. As can be seen, the VAE models outperformed the baseline model at all times.

Table 4.1: Mean squared error for the SVI, Pointwise, and Gridwise approach using the validation set (S&P500) (all values are multiplied by 10^6 and rounded to two decimal points).

Points available	SVI	Pointwise	Gridwise
10	7062.58	25.30	32.30
20	259.06	18.30	18.80
35	18.00	13.50	17.80

Table 4.2: Mean squared error for various indices and their corresponding SVI approach using the surfaces from the year of 2023 (all values are multiplied by 10^6 and rounded to two decimal points).

Points available	SX5E	NDX	UKX	DAX	SX5E_SVI	NDX_SVI	UKX_SVI	DAX_SVI
10	73.80	77.10	24.30	73.90	8082.81	3076.24	62.60	1827.99
20	39.70	40.10	10.50	46.60	397.45	807.00	42.40	89.20
35	33.70	37.40	12.60	42.40	35.30	53.30	40.00	74.00

The following table illustrates the number of days containing static arbitrage when generating volatility surfaces for the validation set with a different number of available points. As can be seen, the percentage of days containing arbitrage is around 1.6%.

Table 4.3: Number of days containing static arbitrage for a total of 499 days of the validation set.

Points Available	Without arbitrage repairing	With arbitrage repairing
35	8	0
10	6	0

Tables 4.4 4.5 showcase the parameters used in the final models. As can be seen, the gridwise approach results in a more complex model with more nodes and twice as many hidden dimensions. The other parameters, except the batch size, have been kept the same for the two different models.

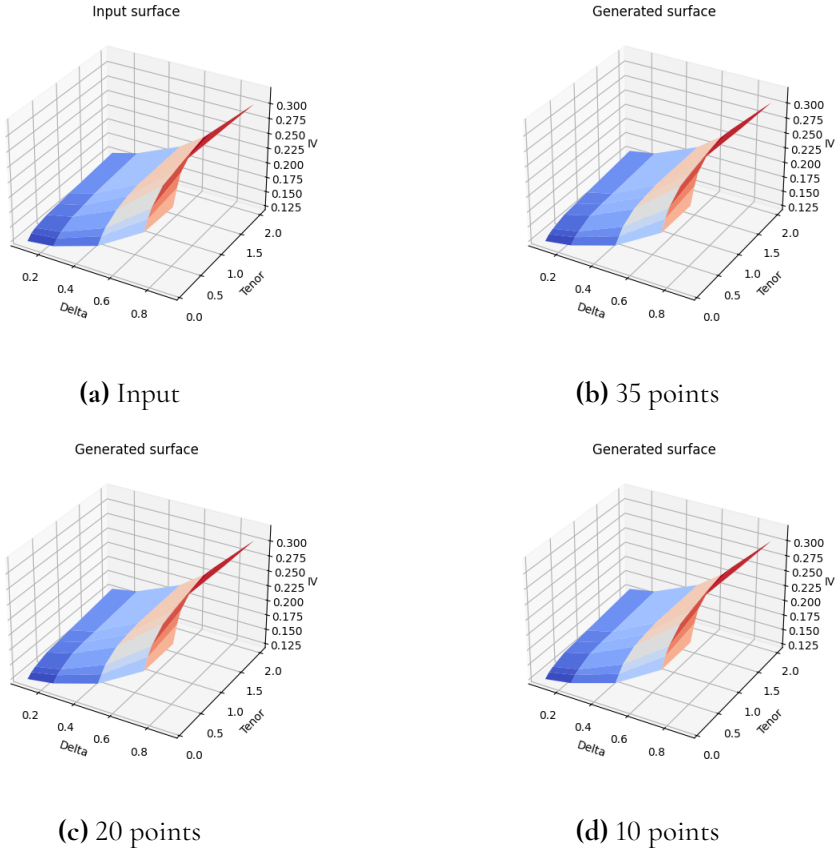
Table 4.4: The pointwise model parameters

Parameter	**Value**
Epochs	44
Latent Space	4
Hidden Dimensions	[223,44]
Learning Rate	0.00042943055075348056
Beta	0.00000105351464768055
Weight Decay	6.756022822706008e-05
Batch Size	59

Table 4.5: The gridwise model parameters

Parameter	**Value**
Epochs	44
Latent Space	4
Hidden Dimensions	[84]
Learning Rate	0.00042943055075348056
Beta	0.00000105351464768055
Weight Decay	6.756022822706008e-05
Batch Size	3

The following plots illustrate 4 different volatility surfaces. Figure 4.1a represents a commonly found volatility surface, Figure 4.5a represents a transition period between a high and a low volatility regime, and Figures 4.13a 4.9a showcase a volatility surface during a high volatility period such as during the covid pandemic. Included are the input surfaces that are fed into the encoder, the 3 volatility surfaces generated when giving the gridwise model access to 10,20 and 35 points in the grid, and the volatility smiles for each tenor separately with the bid-ask spread included. These plots showcase our model's ability to generate different types of volatility surfaces accurately.



(a) Input

(b) 35 points

(c) 20 points

(d) 10 points

Figure 4.1: Visualization of a common volatility surface. The input surface is first visualized and the following surfaces are generated by the VAE when allowed access to 35, 20, and 10 points. (Number 161 of validation set).

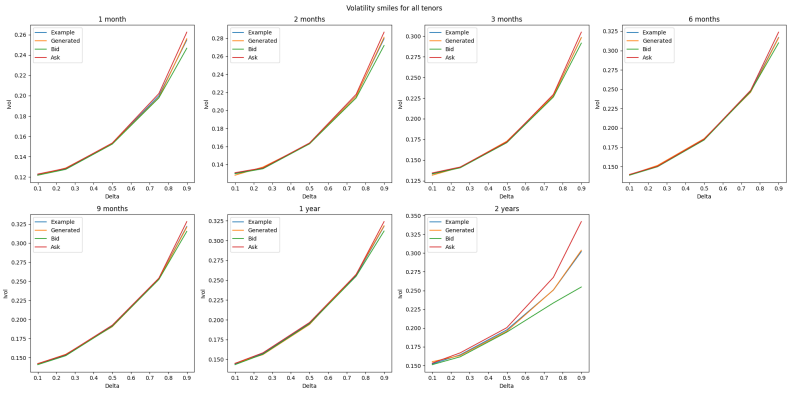


Figure 4.2: Visualization of volatility smiles generated with 35 points along with associated bid-ask spreads. (Number 161 of validation set)

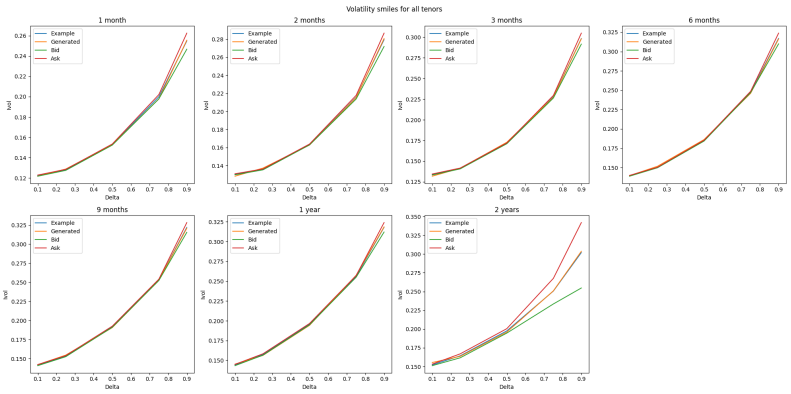


Figure 4.3: Visualization of volatility smiles generated with 20 points along with associated bid-ask spreads. (Number 161 of validation set)

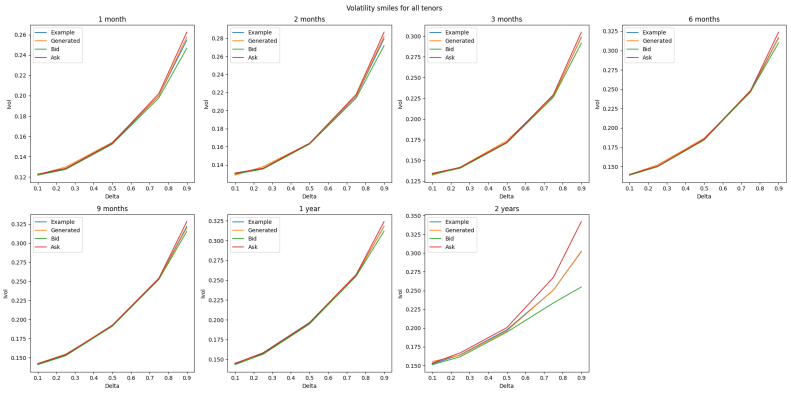
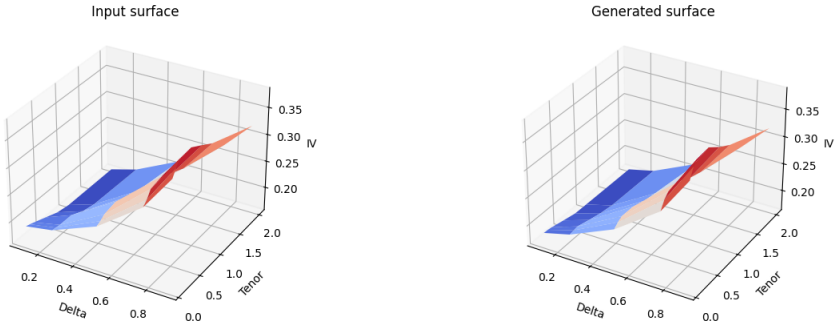
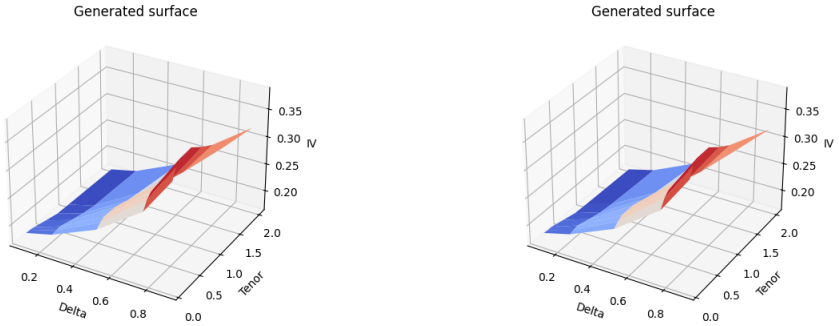


Figure 4.4: Visualization of volatility smiles generated with 10 points along with associated bid-ask spreads. (Number 161 of validation set)



(a) Input

(b) 35 points



(c) 20 points

(d) 10 points

Figure 4.5: Visualization of a volatility surface present during transitions between low and high volatility regimes. The input surface is first visualized and the following surfaces are generated by the VAE when allowed access to 35, 20, and 10 points. (Number 166 of validation set).

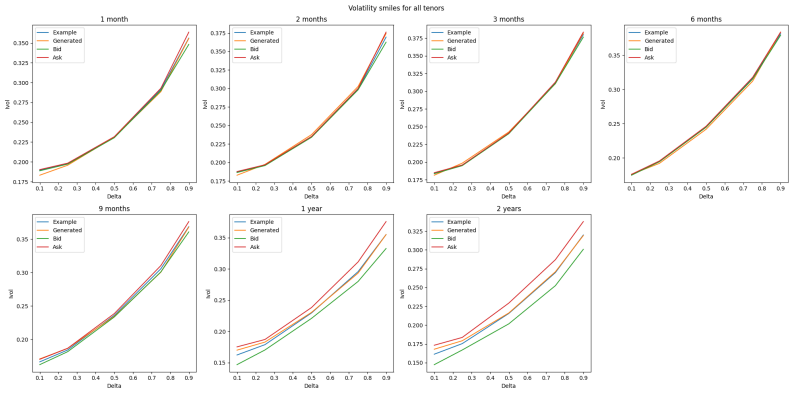


Figure 4.6: Visualization of volatility smiles generated with 35 points along with associated bid-ask spreads. (Number 166 of validation set)

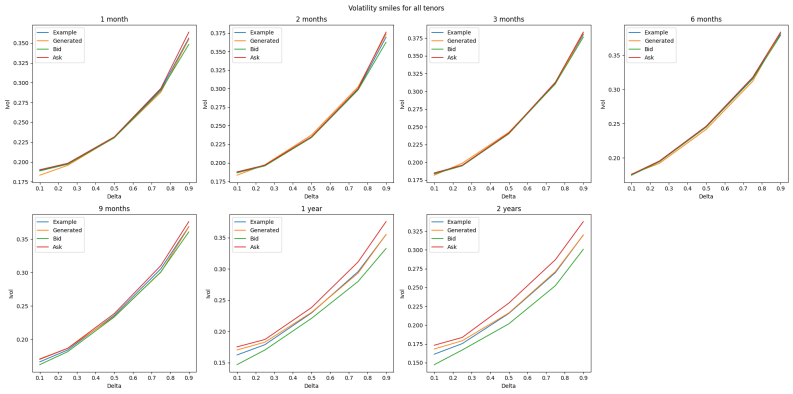


Figure 4.7: Visualization of volatility smiles generated with 20 points along with associated bid-ask spreads. (Number 166 of validation set)

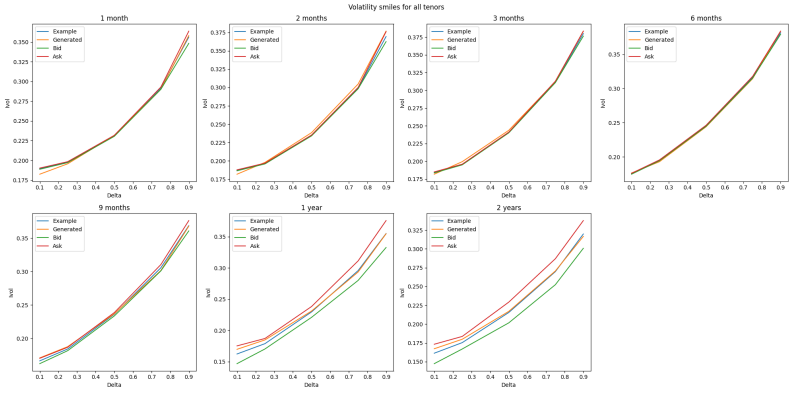
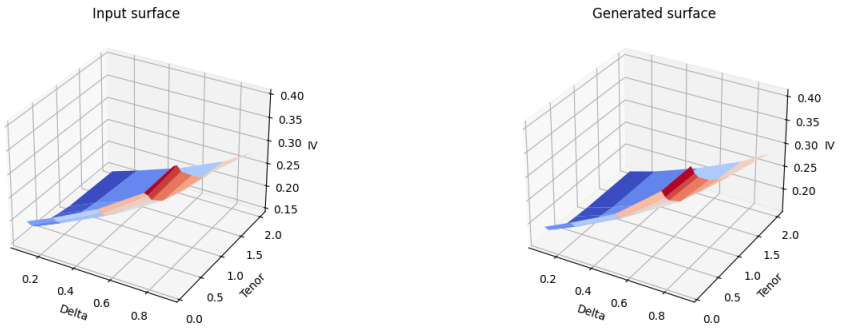
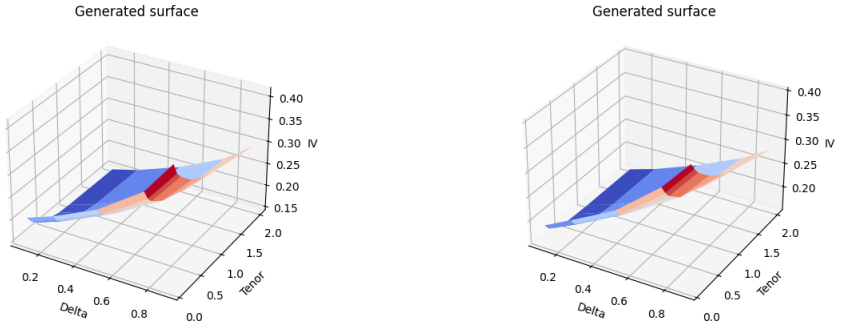


Figure 4.8: Visualization of volatility smiles generated with 10 points along with associated bid-ask spreads. (Number 166 of validation set)



(a) Input **(b)** 35 points



(c) 20 points **(d)** 10 points

Figure 4.9: Visualization of a volatility surface during a high volatility regime. The input surface is first visualized and the following surfaces are generated by the VAE when allowed access to 35, 20, and 10 points. (Number 275 of validation set).

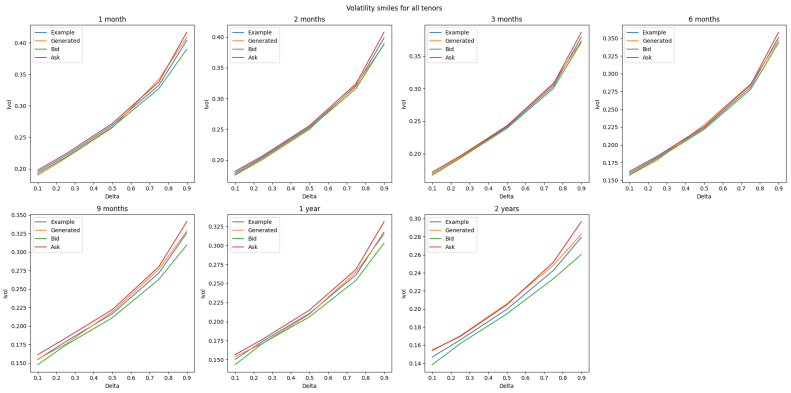


Figure 4.10: Visualization of volatility smiles generated with 35 points along with associated bid-ask spreads. (Number 275 of validation set)

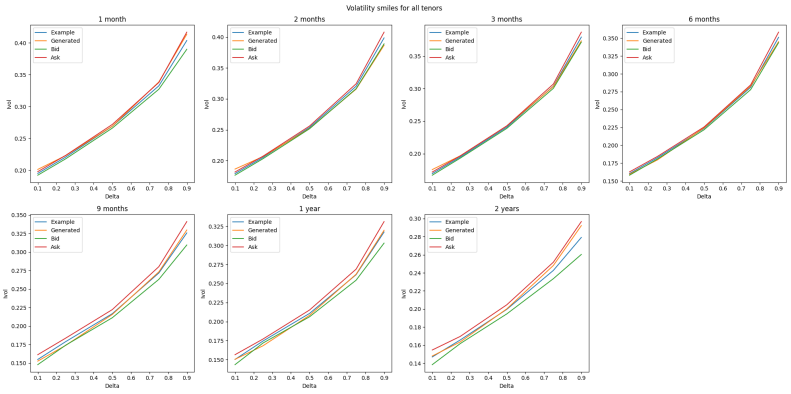


Figure 4.11: Visualization of volatility smiles generated with 20 points along with associated bid-ask spreads. (Number 275 of validation set)

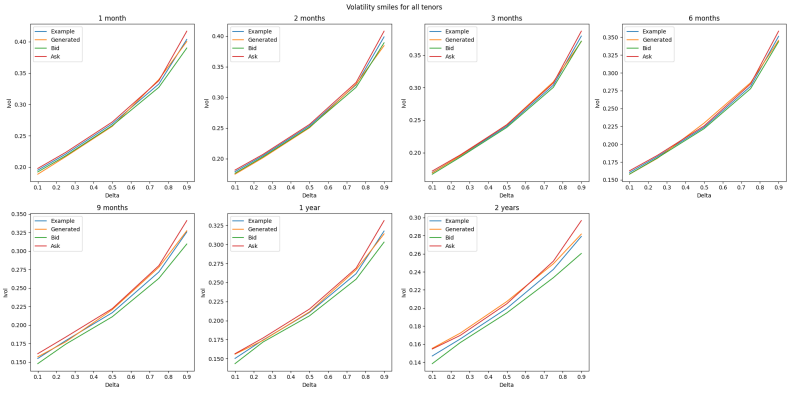
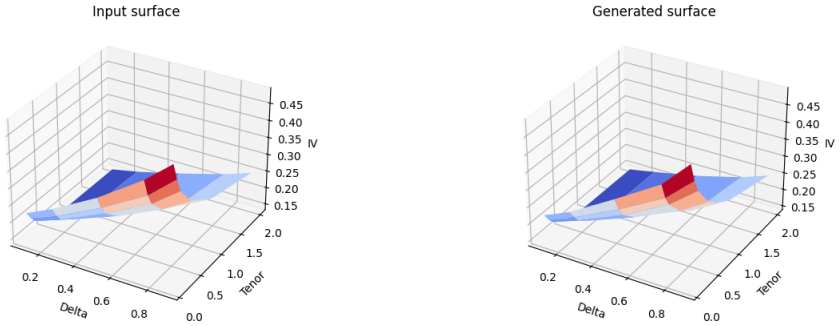
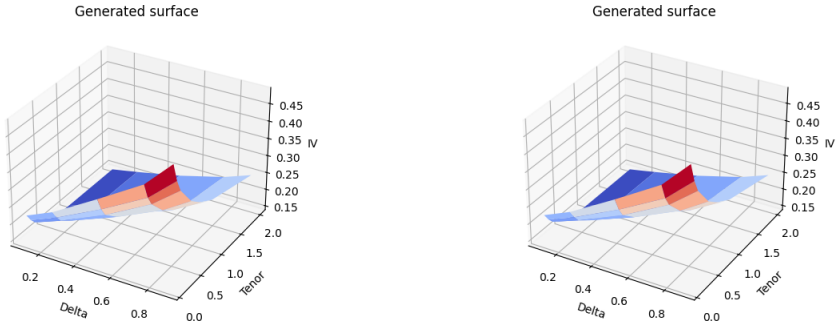


Figure 4.12: Visualization of volatility smiles generated with 10 points along with associated bid-ask spreads. (Number 275 of validation set)



(a) Input

(b) 35 points



(c) 20 points

(d) 10 points

Figure 4.13: Visualization of a volatility surface present during a very high volatility regime such as the covid crisis. The input surface is first visualized and the following surfaces are generated by the VAE when allowed access to 35, 20, and 10 points. (Number 411 of validation set).

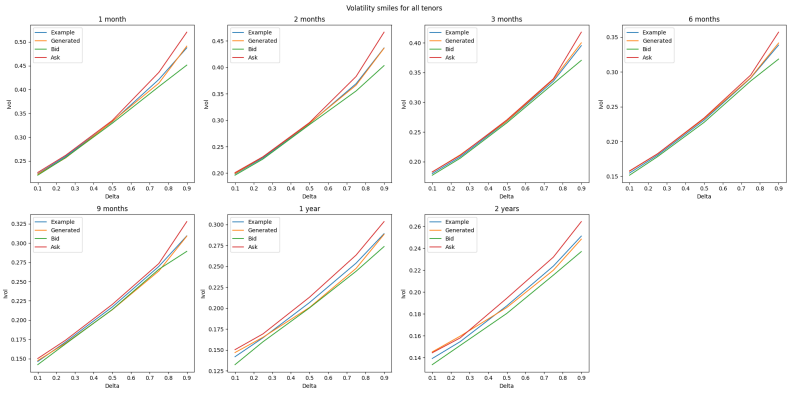


Figure 4.14: Visualization of volatility smiles generated with 35 points along with associated bid-ask spreads. (Number 411 of validation set)

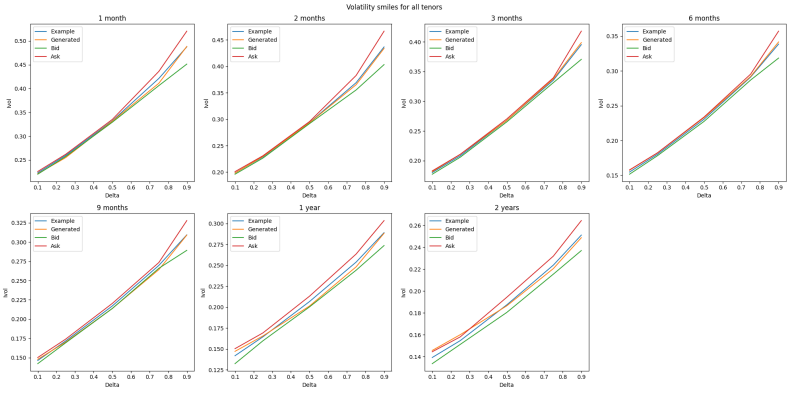


Figure 4.15: Visualization of volatility smiles generated with 20 points along with associated bid-ask spreads. (Number 411 of validation set)

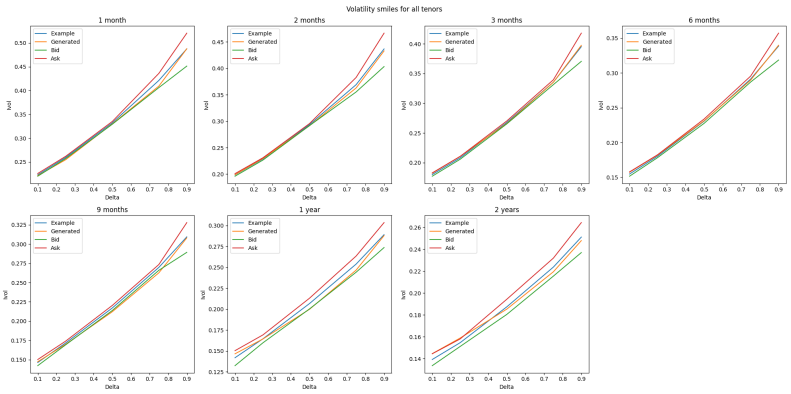
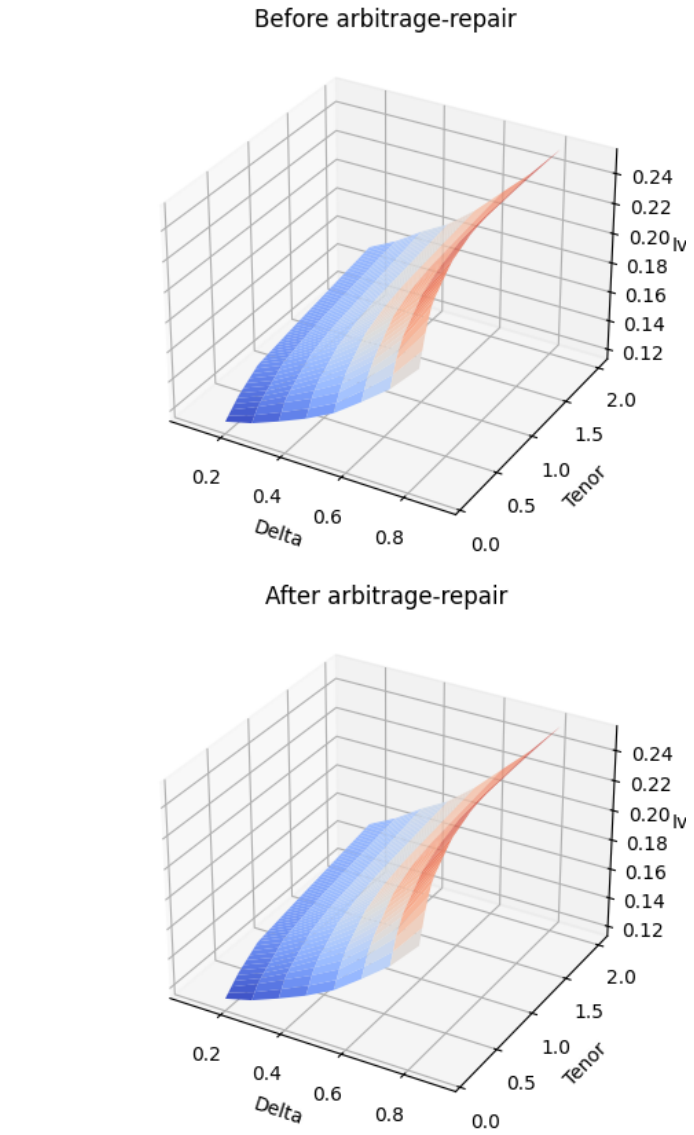


Figure 4.16: Visualization of volatility smiles generated with 10 points along with associated bid-ask spreads. (Number 411 of validation set)

The following figure displays a generated volatility surface containing arbitrage and how it changes after being exposed to the arbitrage-repair function.




```
Before arbsmoothing
Number of violations to non-negative outright price:                  0/7
Number of violations to non-negative and unit-bounded vertical spread: 0/420
Number of violations to non-negative butterfly spread:                3/406
Number of violations to non-negative calendar (horizontal) spread:    0/0
Number of violations to non-negative calendar vertical spread:        0/1211
Number of violations to non-negative calendar butterfly spread:       0/9077
After arbsmoothing
Number of violations to non-negative outright price:                  0/7
Number of violations to non-negative and unit-bounded vertical spread: 0/420
Number of violations to non-negative butterfly spread:                0/406
Number of violations to non-negative calendar (horizontal) spread:    0/0
Number of violations to non-negative calendar vertical spread:        0/1211
Number of violations to non-negative calendar butterfly spread:       0/9077
```

Figure 4.17: Visualization of how the arbitrage-repair function changes the volatility surface. The first image contains 3 instances of butterfly arbitrage, and the second image displays the volatility surface after removing the arbitrage.

4.1.1 Latent Space Visualization

Our model's latent space has 4 dimensions so to visualize it we project to a 2D space using Principal Component Analysis (PCA). The following Figure 4.18 shows where samples in the validation set are projected on a 2D surface and the Figure 4.19 visualizes the volatility surfaces for different values of X and Y.

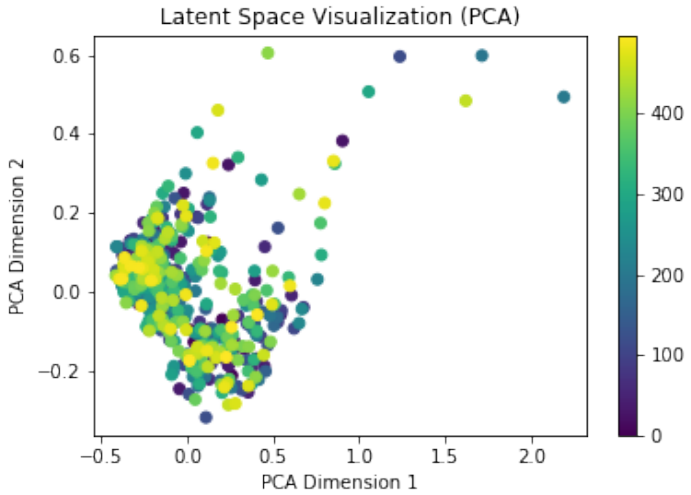


Figure 4.18: A visualization of our latent vectors from our validation set projected to a 2D space using PCA. Each point represents a different type of volatility surface depending on the two dimensions.

4.2 Spot Model Results

The results from the spot model can be seen below. In Table 4.6 we can see the final hyperparameters for the spot model. As can be seen, the spot model requires the highest amount of nodes of all models. The validation score of the spot model can be seen in Table 4.7 and can be observed to be higher than the validation score of the pointwise model. Furthermore, visualizations of how the spot model predicts changes to the volatility surface in regard to large changes in spot price can be seen in Figures 4.21 4.20.

Table 4.6: The spot model parameters

Parameter	Value
Epochs	76
Latent Space	4
Hidden Dimensions	[223,122]
Learning Rate	0.00082943055075348056
Beta	0
Weight Decay	6.756022822706008e-05
Batch Size	105

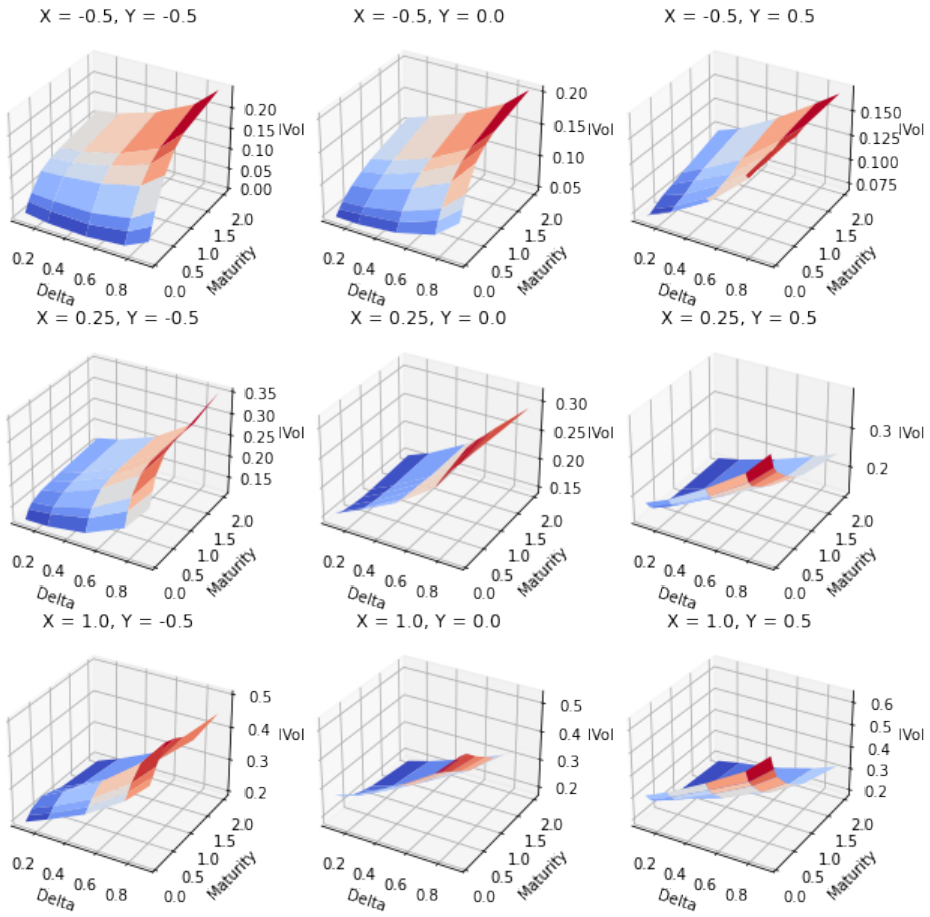


Figure 4.19: Visualization of surfaces produced by the 2D projected latent vectors.

Table 4.7: Mean squared error for the spot model using the validation set (the value is multiplied by 10^6 and rounded to two decimal points).

Validation score	69.88

Table 4.8: Number of days containing static arbitrage for a total of 499 days of the validation set.

Points Available	Without arbitrage repairing	With arbitrage repairing
35	18	0

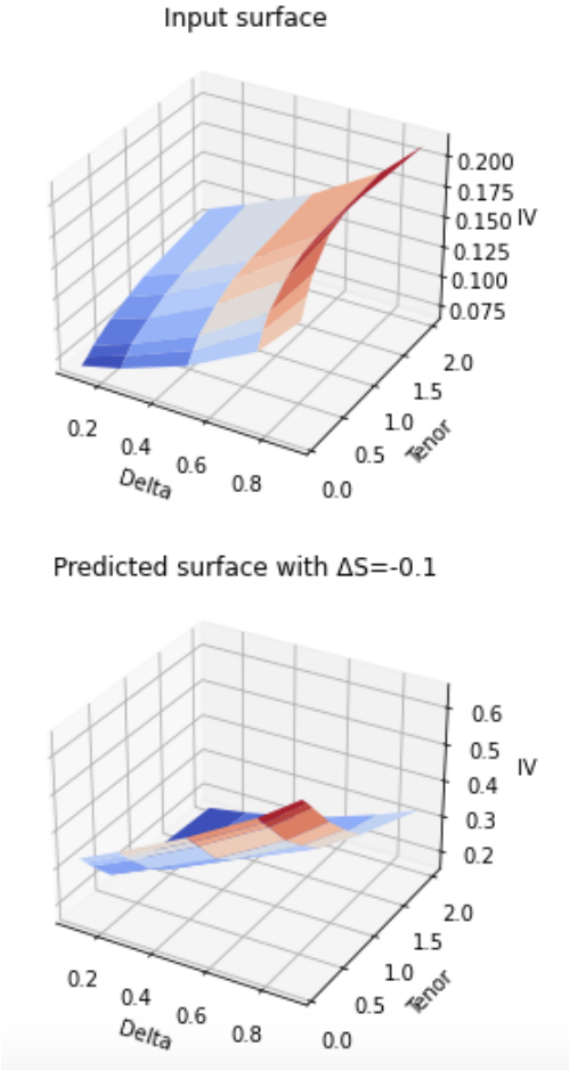


Figure 4.20: The input surface of a normal day and the spot model's predicted surface with a change in the spot price by -10%

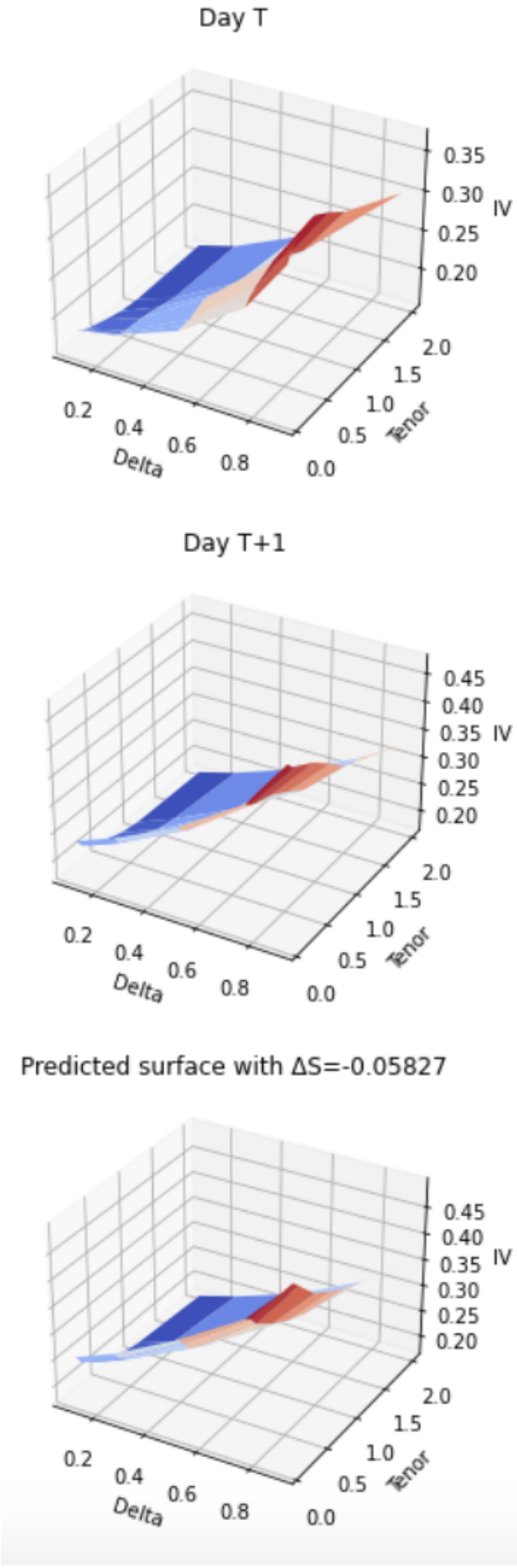


Figure 4.21: The surface of day T and day T+1 with the largest negative ΔS found in our validation set and the spot model's predicted surface.

Chapter 5
Discussion

5.1 Data

The data handling aspect of this project was one of the most time-consuming and difficult parts. All previously published work done on modeling implied volatilities with variational autoencoders had been done on FX markets where the options are quoted in volatilities for fixed deltas. This leads to the first issue when modeling index equities in the same way as other papers had modeled FX rates. Without any alterations to the data, we would not be able to provide our model with a grid of fixed delta values since an option with the delta needed for our model might not exist for that specific day. We then had two choices, either alter our model to encode arbitrary delta and tenor values or interpolate the data to match our desired grid. The first approach would result in a completely different VAE structure which would require additional information in the input layer. Due to this, we felt that the second approach was more true to the nature of VAEs. Interpolation of option prices is a delicate process as well, which might induce arbitrage if done improperly, however, we felt that it was necessary to obtain the best possible training data for our model. When interpolating we made sure to do it in an arbitrage-free way as seen in Section 3.4. Another issue with the data was the inconsistency in the range of strikes and tenors quoted each day. Since we wanted a consistent grid as input we would either have to filter out the majority of our data or settle for a smaller grid. We decided to do a combination of both to make sure our grid was large enough to be useful in practice. In the beginning of our project, we included one week tenors as well, but since only around half of our dataset actually had options with tenors shorter than a month, the VAE was not able to process it well. Due to this, we shortened our grid to only include tenors longer or equal to one month. For the same reason, we chose 2 years as the longest tenor. We had the same issue regarding delta values. Some days would have quotes spanning the entire space from 0 to 1 while some days would cover a much smaller space. The majority of days would however contain options with delta values between 0.1-0.9 which is why we decided to go for that range.

After training our VAE with the data extracted through this approach we noticed that our produced volatility surfaces contained arbitrage. We determined that this could be due to two things. Either our in-data had arbitrage present causing the VAE to learn volatility surfaces containing arbitrage, or our VAE was inaccurate and that inaccuracy was causing arbitrage. Through plotting generated volatility surfaces and observing that they matched the input data almost perfectly we concluded that the problem must lie, at least partly, with the in-data. We then ran our arbitrage testing algorithms on the mid prices from our option quotes and discovered that almost every day contained arbitrage, in line with the findings of [9]. This would largely be caused by quotes with large spreads, so we attempted to filter out the data with too wide bid-ask spreads. This had the unintended effect of reducing our grid size even more since the options trading at delta values of around 0.9 are rather illiquid, and most arbitrage was caused by options with deltas close to 0.9. Because of this, we decided to run the arbitrage repair on our mid-prices before interpolating and creating the grid. Our new input grids would have the same size as before but be arbitrage-free as a result of this. Since the VAE is trained to reproduce the input, assuming the input data is arbitrage-free, the output should also be arbitrage-free if perfect accuracy is assumed.

5.2 Results

As seen in Table 4.1 the pointwise approach outperforms both the baseline SVI model and the gridwise model. This is especially noticeable when generating surfaces with incomplete information as seen in the MSE loss when only allowing the model access to 10 or 20 points. It seems natural that both of the VAE models would heavily outperform the SVI model with fewer points available since all surfaces produced by the VAE models are similar to surfaces that have been seen when training on a 35-point grid while the SVI model fits the parameters from scratch solely based on the 10 points available. We can observe from Table 4.1 that both VAE models seem to learn the patterns of volatility surfaces excellently, even outperforming the standard SVI model. In Table 4.1 we see that both the pointwise and gridwise models improved with more points available, as expected. The results obtained show that the pointwise model outperformed the gridwise model, however, we believe that with even better hyperparameters the gridwise model should be able to match the result of our pointwise model in terms of MSE loss. The reason why we chose the pointwise model as the optimal model, was not only due to the lower MSE loss but rather its flexibility in training, and generation of points. With the pointwise model, we can generate implied volatilities for any delta and maturity inside the grid while the gridwise is only able to generate volatilities for values matching the standard grid exactly. In practice, investors might not have access to deltas and tenors that correspond to our input grid, in which case the gridwise model would be useless unless investors interpolate in real time to produce values suited for the model. A gridwise model makes more sense in the FX markets where options are quoted in fixed deltas, therefore in the more chaotic equity markets, the adaptability of the pointwise model is preferred.

When testing the model's capabilities on other indices we can see in Table 4.2 that it per-

forms surprisingly well considering it is unseen data for the VAE. The data of the indices other than the SPX are also in a different form. Since we do not interpolate to create a perfect grid, the surfaces contain 35 points of arbitrary tenor and delta values. This allows us to test our pointwise model's accuracy when generating implied volatilities not exactly on the grid. A satisfactory outcome in generating surfaces from arbitrary points demonstrates a practical application for our model, as it reflects the real-world scenario where investors may encounter limited and inconsistent option prices when estimating implied volatilities for exotic options. Due to the nature of the different indices, the MSE loss can not be compared one-to-one with the MSE loss from the SPX. A varying degree of bid-ask spreads depending on the liquidity of the chosen index may give rise to unrealistic volatility surfaces which our model has not been trained to generate. The data quality of the different indices was also wildly inconsistent. The OMX for example had periods where we only had data on as little as 5 options per day causing us to discard results on the index completely. Therefore it makes more sense to compare the MSE loss with the baseline SVI model since the baseline model gives an indication on how difficult the volatility surfaces were to model due to incomplete data or illiquid options. Comparing these models we can see that UKX was the easiest index for our model. When plotting the surfaces for the UKX we noticed that the surfaces for every day looked very similar, which we believe is the reason for an MSE loss even lower than for the SPX in Table 4.1. We believe this is also the reason why the SVI model performed so well using only 10 points. A strange result obtained in Table 4.2 was that for the UKX, the MSE loss for 20 points was lower than for 35 points. We reran this experiment numerous times to verify the legitimacy of the results and obtained the same result every time. We struggled to find a reason to explain the results, however, we think this has to do with the similarities of the surfaces for each day, and that with fewer points we create a simpler surface. In Table 4.2 we can see that the pointwise model outperforms the SVI model in all metrics proving its usefulness.

Looking at Figures 4.1b 4.5b 4.13b 4.9b we can see several different volatility surfaced modeled accurately. Different volatility surfaces provide different challenges for our model. In Figure 4.1b we can see the most common volatility surface found in our dataset. Due to this, the model can generate extremely accurate implied volatilities for similar surfaces. Figure 4.5b shows a volatility surface highlighting the transition between a low and a high volatility period for the underlying asset. These surfaces are quite rare in the dataset leading to a higher degree of error when generating them with our VAE. The rarest volatility surfaces can be seen in Figures 4.13b 4.9b which showcase a period of extreme volatility such as during the covid crisis. These volatility surfaces typically only last for a few days at a time and are not present every year leading to only a few surfaces similar to these present in the training data. Even with the limited amount of training done with these kinds of surfaces, the VAE is still able to produce accurate recreations, although not quite as accurate as the standard volatility surface. As seen in Figure 4.14 we can generate implied volatilities mostly within the bid-ask spread. During testing, we found that the accuracy of the VAE was the worst for the one month tenor and the two-year tenor. In the case of the one month tenor, we believe that the difficulty can be explained by the higher levels of volatility in options closer to expiry. In Figure 4.10 we can see that the bid-ask spread becomes larger with longer tenors leading to more uncertainty in the mid-price. This may cause inconsistencies in the training data which seems to be the reason for our model struggling more with the accuracy of the

two-year tenor than the shorter ones. This is however compensated by the bid-ask spread being large, so even with less accuracy towards mid-price our generated implied volatilities are still within the spread. Another observation is that the bid-ask spread is the largest at delta values around 0.9 as seen in Figure 4.14. This may also confuse our model due to the vast difference in mid prices at delta 0.9 for otherwise similar volatility surfaces. Two volatility surfaces should perhaps in theory be more similar than what their mid prices reflect due to illiquid option prices. As seen in Figures 4.16 4.15 4.14 we can also see that the model struggles more with keeping the generated volatilities within the bid-ask spread with fewer points available caused by higher inaccuracies as showcased previously.

5.2.1 Latent Space Visualization

When visualizing the latent space we see many interesting things, firstly visualizing the latent space after PCA in Figure 4.18 we see where the different surfaces are represented in the latent space. This combined with plotting the surfaces corresponding to values in the visualized latent space as in Figure 4.19 shows us where certain shapes are projected in the latent space. We can see in Figure 4.18 that most points are very close together, however, some points are outliers with high X and Y values. When plotting these values we see that those surfaces correspond to early covid surfaces with very high volatility on short maturities as seen where X=1 and Y=0.5 in Figure 4.19. In Figure 4.19 we also see a wide representation of the different shapes of volatility surfaces produced by our model. Analysing the values of X and Y and their impact on the surface we note that a larger value of X seems to have the biggest impact on the overall volatility whereas a change in Y has more of an impact on how the volatility changes with tenor. With X=1 and the Y changing from -0.5 to 0.5 we go from low volatility on shorter tenors and higher volatility on longer tenors to high volatility on short tenors and lower volatility on higher tenors. This causes a combination of higher X and Y values to result in plots resembling the volatility surface during the early days of covid, since this combination gives high overall volatility with the shortest tenors the most volatile ones.

5.2.2 Arbitrage

Ensuring arbitrage-free surfaces has been one of the most difficult and time-consuming parts of this project. Initially, we did not take any action to remove arbitrage and only tested for it after generating the volatility surfaces. We then discovered that our surfaces contained arbitrage by a rate of 5% and we had to figure out a way of removing it. Because of this, we decided to investigate what was causing this arbitrage. Most of the option combinations resulting in arbitrage opportunities included an option with a delta value of around 0.9 where the bid-ask spread was large as previously stated. The initial hypothesis posited that if the training data were arbitrage-free, then the VAE would generate arbitrage-free surfaces provided its output accurately mirrored the input, as discussed in [3]. We then decided to use the arbitrage repair algorithm on the initial data to make sure that the input to the VAE was arbitrage-free. Looking at Table 4.3 our theory seems to be partly correct. While the number of arbitrage days generated by the VAE perhaps is negligible from a statistical standpoint, this would still be a problem for traders looking to implement the model. At this point, we had two options on how to remove the arbitrage. Either we introduce another loss function

to the training part of the VAE or we remove the arbitrage after producing the volatility surfaces. Since Jim Gatheral and Antoine Jacquier [12] work with mathematic formulas such as the SVI model to create volatility surfaces they can guarantee their models to be free of static arbitrage in the continuous case. This is more difficult for us to achieve, and a one-to-one implementation of their work is not possible. Punishing arbitrage in the loss function is very difficult due to the current structure of our VAE. Had we chosen to generate the parameters of an SSVI model instead of directly generating volatility surfaces we could have applied the static arbitrage conditions found in [12]. Since our model only produces 35 discrete points in the training section, we would only be able to ensure that our model was arbitrage-free in a discrete space and not in a continuous. We would also require many more points than 35 to test for arbitrage in a realistic space. In contrast to the SVI model, there is no easy way of ensuring an arbitrage-free model in the continuous space. There are also many different ways we could have tested for arbitrage. We decided to generate option prices for every strike interval of 25. Surprisingly, when calculating implied volatilities for each strike within a 100-strike interval, we found no instances of arbitrage whatsoever. However, using intervals of 10 might have led to even more instances of arbitrage. Since an interval of 25 represents 0.5% of the current spot price we thought that size to be reasonable.
An interesting observation is that we seem to generate fewer days containing arbitrage when feeding the model only 10 points as seen in Table 4.3. This might be due to the model generating a more standardized volatility surface learned from the training data, and since the training data is arbitrage-free, the more standardized volatility surfaces learned will be as well. Since only 1.6% of our days contained arbitrage, we concluded that the best solution was to use the arbitrage-repair function on surfaces after generating them. By doing this we remove all arbitrage in the discrete space. After repairing the surface we investigated which option prices the algorithm chose to change. We found that usually, it is enough to change the 0.9 delta options by a very small amount which barely changes the surface as seen in Figure 4.17.

5.2.3 Spot Model

As we can see in Table 4.7 the validation score (MSE) for the best spot model was 69.88. This compared to 13.5 for the VAE model as seen in Table 4.1 suggests that creating a volatility surface using the previous day and a change in spot price is a much harder problem to solve. This is especially true since the structure of the networks and the parameters were very similar as seen in Tables 4.4 4.6. A big difference between the models, however, is that the spot model has the beta set to 0. This is because we are now only interested in getting the reconstruction error as low as possible. Since the purpose of the spot model is to make accurate predictions and not create surfaces from fewer points, we thought that adding any variation σ to the optimal solution μ found would result in a worse prediction.

The result of 69.88 shows us that there is a significant correlation between the spot price and the volatility surface and that by only using a change in spot price we can accurately predict what tomorrow's volatility surface will look like. An example of this can be seen in Figure 4.21 where we display the predicted surface on the biggest ΔS. This allows our model to be used as a tool to analyze how volatility surfaces respond to changes in the underlying spot price. When testing our model we noticed something very interesting as seen in Figure 4.20. We tested a very common volatility surface as input and chose a change in the spot price

of -0.1 (-10%). As an output, we got a volatility surface resembling a surface during the first days of covid. Through analyzing further combinations of volatility surfaces and changes in spot price we can draw a similar conclusion to the research in [7] where the general level of a volatility surface and the change in spot price were deemed negatively correlated for the most part. A negative change in spot price seems to produce a volatility surface with a shape closer to the shapes seen during the covid era with a higher general level of implied volatility. Meanwhile, starting with a high volatility level surface as seen in Figure 4.13a and generating a new surface for a large positive change in spot price decreases the general volatility level and the shape of the surface reverts to the shape seen in Figure 4.1a. Looking at Figures 4.13a 4.1a we can also observe that the largest changes occur for options of high delta and short tenors as previously found in[7]. This further highlights the potential use of our model as a risk analysis tool not only for predicting the general level of volatility but also for the specific shape of the volatility surface.

A practical and intriguing application of our two models is their integration to generate a volatility surface on days with limited available options. This surface can then be input into our spot model, allowing us to visualize how tomorrow's grid will look with different spot price changes.

In regards to arbitrage, we can see in Table 4.8 that we had 18 days of surfaces that featured arbitrage. Comparing this to that of the VAE model with 35 points containing 8 days as seen in Table 4.3. With the spot model having a bigger error than the VAE model we would expect it to also generate more surfaces containing arbitrage. After arbitrage repairing we could see in Table 4.8 that none of the surfaces contained arbitrage.

We thought that a more interesting application of the model would be to model days with a big change in the spot price and therefore we looked at weighing the points based on the price change, the idea being that it would be better at accurately predicting bigger spot changes. This turned out to be the case but it came at the cost of lower accuracy on smaller changes in the spot price. Therefore we ultimately decided against using a weighting scheme.

5.3 The Impact of the Model's Parameters on the Result

In the process of finding the best model, we experimented with parameters and some results we came up with are worth discussing. To begin with, the problem of creating volatility surfaces seems fairly easy to solve. We draw that conclusion because our best gridwise model only features one layer as seen in Table 4.5. Adding in the extra information (delta and maturity) in the latent space is what made it a harder problem and as a result, we think that is why our best pointwise model used 2 layers instead of 1, see Table 4.4. When experimenting with even more layers we found the model starting to overfit, performing worse on the validation and test set.

Another parameter we experimented with was the latent space. The dimension of the latent space is a tradeoff, the bigger the latent space, in theory, the better accuracy since patterns can be separated by more dimensions. However, in the case of a VAE when we want to draw and generate samples, adding dimensions makes it harder since it might result in worse generalization. We therefore chose to have our latent space set at 4, since fewer dimensions gave us worse accuracy and more dimensions resulted in the model being more

overfitted. As seen in Table 4.5 the batch size for the grid-based model is 3 and in Table 4.4 we see that the batch size for the pointwise model is 59. This significant difference in batch size can be easily explained. In the pointwise approach the batch size signifies the amount of points used from different days, whereas in the case of the gridwise approach, the batches are done in days with 35 points each. So a batch size of 3 in the gridwise model correlates to a batch size of 105 for the pointwise model. Therefore we can see that the pointwise model used about half the amount of points per batch compared to the gridwise model.

5.3.1 The Beta Tradeoff

The parameter with the biggest impact on our result, both on the training and validation set was beta. The beta weighs the importance of the KLD Loss compared to our reconstruction loss. With a higher beta, we would find ourselves with a bigger reconstruction error and a relatively non-existent KLD loss. With a lower beta, we instead found the network working more like an autoencoder, where we would get much better reconstruction error but at the cost of a higher KLD loss, and as a result, a harder time sampling the latent space. Since the patterns would not end up in a unit normal distribution with a higher KLD loss, but rather in an arbitrary distribution decided during training. When plotting surfaces generated by models with lower betas we saw that it became better at recreating our most common surfaces however struggled with less common surfaces such as the surface visualized in Figure 4.13b. This is likely due to the MSE loss being minimized by the model performing well on the more common days. Higher beta resulted in the model becoming better at the less common surfaces at the cost of accuracy. We were simply in a tradeoff with no trivial solution. We believe however that the beta obtained after running hyperparameter tuning allows the model to perform well in both aspects. It gives us a low reconstruction error as seen in Table 4.1 but also can be easily visualized as in Figure 4.19. We can also see that after PCA we get a pleasant distribution of points for generating samples as seen in Figure 4.18.

5.4 Model Limitation & Future Work

Above we have discussed some of our model's capabilities and results, however, there are also some limitations to our models. One negative aspect of the model is that it does not always generate prices within the bid-ask as can be seen for delta 0.1 for the one month tenor in Figure 4.6, this is due to us not including the bid-ask anywhere in the loss-function. To enhance the VAE in the future, we consider incorporating a penalty term to discourage the generation of numbers outside the bid-ask spread. Another limitation of our VAE is that it does not perform well at extrapolating deltas and maturities outside of the delta and tenor space it has been trained on. To improve this in the future we would require additional data so that we could train the model on shorter maturity dates and delta values outside 0.1-0.9. An issue with our model is that it still is capable of producing volatility surfaces containing arbitrage. Since we cannot guarantee arbitrage-free volatility surfaces in the continuous space, further research would be required to find a solution. While the results regarding the other indices outperform the SVI model, further training on more indices than the SPX would lead to the model learning an even wider variety of volatility surfaces and most likely performing even better on other indices. This would require cleaning the data for the other indices in the same

way as we did for the SPX and including them in the training data. This would however result in a very large training set and it would take a long time to optimize the hyperparameters for this new dataset. We decided to omit them from our dataset partly due to the low quality of data and our computing power available. All work was done on an Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz processor, 8 GB RAM, and an NVIDIA GeForce GTX 1650. With access to more powerful computing power, we would most likely have found better hyperparameters for our model as well. Another idea for future work is that instead of interpolating a grid and feeding it to the encoder as we experimented with, a different approach would be to tweak the encoder to be able to handle arbitrary quotes to preserve our raw data and be able to train it on more points to hopefully get better results.

Chapter 6
Conclusion

In conclusion, our exploration into modeling implied volatilities with Variational Autoencoders (VAEs) has yielded promising results and insights. We have demonstrated the use of VAEs in constructing volatility surfaces in the equities market, particularly focusing on the SPX. Despite the inherent challenges posed by the equities market's volatility dynamics and data irregularities, our models have shown promising performances in generating accurate surfaces, even with limited information. In addition, our model also showed promising results extending to other indices, proving the robustness of our model and generalization capabilities beyond the training data.

One notable achievement of our study is the successful adaptation of VAEs from previously researched usage in FX markets to the context of index equities. Through comprehensive data processing and model adjustments, we have overcome challenges such as the absence of fixed delta quoting in equities options, ensuring that our models are suited to the characteristics of the index options market.

In addition to volatility surface generation, we have explored the predictive capabilities of our models in relation to changes in spot prices. The spot model, trained to predict volatility surface changes based on spot price movements, has shown promising results. By utilizing historical data and spot price changes, our model is capable of providing insights into how volatility surfaces may evolve in response to market movements.

Exploring the latent space of our model has allowed us to explore the complex space of volatility surfaces through only 4 variables. This has provided insight to how our model groups the different kinds of volatility surfaces and provides a deeper understanding of the underlying patterns and structures driving volatility surfaces.

While our models have achieved a satisfactory level of accuracy and a certain degree of arbitrage-free generation, there remain areas for improvement and future research. Enhancements such as incorporating bid-ask spread constraints and extending the training data could further enhance the robustness and practical utility of our models.

In summary, our study highlights the potential of the Variational Autoencoder as a powerful tool for modeling and predicting implied volatilities in equities markets. By addressing

challenges, refining methodologies, and leveraging the insights gained from this research, we aim to contribute to advancements in the use of neural networks in quantitative finance.

References

[1] Damien Ackerer, Natasa Tagasovska, and Thibault Vatter. Deep smoothing of the implied volatility surface. *NeurIPS*, 2020. https://arxiv.org/pdf/1906.05065.

[2] Christian Bayer, Blanka Horvath, Aitor Muguruza, Benjamin Stemper, and Mehdi Tomas. On deep calibration of (rough) stochastic volatility models. arXiv:1908.08806, 2019. https://arxiv.org/pdf/1908.08806.pdf.

[3] Maxime Bergeron, Nicholas Fung, John Hull, and Zissis Poulos. Variational autoencoders: A hands-off approach to volatility, 2021. https://arxiv.org/pdf/2102.03945.pdf.

[4] Tanvir Bhuiyan, Ariful Hoque, and Thi Le. Analysing implied volatility smirk to predict the us stock market crash during the global financial crisis. *Journal of Innovation and Technology Management and Creativity*, 2023.

[5] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *The Journal of Political Economy*, 81(3):637–654, 1973.

[6] Bloomberg. Spx:ind - bloomberg. https://www.bloomberg.com/quote/SPX:IND, n.d. Accessed: [2024-04-25].

[7] Jay Cao, Jacky Chen, and John Hull. A neural network approach to understanding implied volatility movements. *Quantitative Finance*, 20(9), 2020.

[8] Peter Carr and Dilip B. Madan. A note on sufficient conditions for no arbitrage. *Bloomberg LP/Courant Institute, New York University*, 499:Park Avenue, New York, NY 10022, 2005. Available online: https://www.sciencedirect.com/science/article/abs/pii/S1544612305000413.

[9] Samuel N. Cohen, Christoph Reisinger, and Sheng Wang. Detecting and repairing arbitrage in traded option prices. *Mathematical Institute, University of Oxford*, 2020. https://arxiv.org/abs/2008.09454.

[10] Georgi Dimitroff, Dirk Röder, and Christian P. Fries. Volatility model calibration with convolutional neural networks. 9 2018.

[11] Jim Gatheral and Antoine Jacquier. Convergence of heston to SVI, 2010. https://arxiv.org/pdf/1002.3633.

[12] Jim Gatheral and Antoine Jacquier. Arbitrage-free SVI volatility surfaces, 2013. https://arxiv.org/pdf/1204.0646.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[14] Martin Haugh. *Foundations of Financial Engineering*. Martin Haugh, 2016. IEOR E4706: The Black-Scholes Model.

[15] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes, 2014. https://arxiv.org/pdf/1312.6114.pdf.

[16] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. arXiv:1711.05101, 2019. https://arxiv.org/pdf/1711.05101.pdf.

[17] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, page 3, 2013. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[18] Brian (Xin) Ning, Sebastian Jaimungal, Xiaorong Zhang, and Maxime Bergeron. Arbitrage-free implied volatility surface generation with variational autoencoders, 2022. https://arxiv.org/pdf/2108.04941.pdf.

[19] Mattias Ohlsson and Patrik Edén. *Introduction to Artificial Neural Networks and Deep Learning*. Lund University, Computational Biology and Biological Physics, Department of Astronomy and Theoretical Physics, 2023.

[20] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv:1511.06434, 2016. https://arxiv.org/pdf/1511.06434.pdf.

[21] Kihyuk Sohn, Honglak Lee, and Xin Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, volume 28, pages 3483–3491, 2015.

[22] Yu Zheng, Yongxin Yang, and Bowei Chen. Gated deep neural networks for implied volatility surfaces. 2019.

Appendices

Appendix A
Additional information

Validation Points	Delta Values
10	1M: (0.1, 0.9) 2M: (0.5) 3M: (0.1, 0.9) 6M: (0.75) 9M: (0.1) 1Y: (0.25) 2Y: (0.1, 0.9)
20	1M: (0.1, 0.5, 0.9) 2M: (0.5, 0.9) 3M: (0.1, 0.5, 0.9) 6M: (0.25, 0.75, 0.9) 9M: (0.1, 0.5, 0.9) 1Y: (0.25, 0.75, 0.9) 2Y: (0.1, 0.5, 0.9)

Table A.1: Validation Delta Values

Appendix B

Algorithms

Input: $\gamma(lr), \beta_1, \beta_2$ (betas), θ_0 (params), $f(\theta)$ (objective), ϵ (epsilon), λ (weight decay), amsgrad, maximize
Output: θ_t
Initialize: $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $v_0^{max} \leftarrow 0$;
for $t = 1$ **to** ... **do**
  if *maximize* **then**
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$;
  end
  else
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$;
  end
  $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$;
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$;
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$;
  $m_t' \leftarrow \frac{m_t}{1-\beta_1^t}$;
  $v_t' \leftarrow \frac{v_t}{1-\beta_2^t}$;
  if *amsgrad* **then**
    $\hat{v}_t^{max} \leftarrow \max(\hat{v}_t^{max}, \hat{v}_t)$;
    $\theta_t \leftarrow \theta_t - \frac{\gamma \hat{m}_t}{\sqrt{\hat{v}_t^{max}} + \epsilon}$;
  end
  else
    $\theta_t \leftarrow \theta_t - \frac{\gamma \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$;
  end
end
return θ_t

Algorithm 1: AdamW Optimization Algorithm

Input: Initial guess x_0, tolerance ϵ
Output: Approximation of root $\hat{x}$
$iter \leftarrow 0$;
while $|f(x_{iter})| > \epsilon$ **do**
    Compute $f(x_{\text{iter}})$ and $f'(x_{\text{iter}})$;
    $x_{\text{iter}+1} \leftarrow x_{\text{iter}} - \frac{f(x_{\text{iter}})}{f'(x_{\text{iter}})}$;
    $iter \leftarrow iter + 1$;
end
return $\hat{x} = x_{iter}$

Algorithm 2: Newton-Raphson Method

Input: Initial simplex S_0, tolerance tol
Output: Approximation of minimum
$S \leftarrow S_0$;
while $\sigma(S) > tol$ **do**
    $x \leftarrow x_{n+1}$;
    $x_r \leftarrow x(\rho, n+1)$; // Reflect
    $f_r \leftarrow f(x_r)$;
    if $f_r < f_1$ **then**
        $x_e \leftarrow x(\rho\chi, n+1)$; // Expand
        $f_e \leftarrow f(x_e)$;
        if $f_e < f_r$ **then**
            Accept x_e ;
        else
            Accept x_r ;
        end
    else if $f_1 \leq f_r < f_n$ **then**
        Accept x_r ;
    end
    else if $f_n \leq f_r < f_{n+1}$ **then**
        $x_c \leftarrow x(\rho\gamma, n+1)$; // Outside contraction
        $f_c \leftarrow f(x_c)$;
        if $f_c < f_r$ **then**
            Accept x_c ;
        else
            Compute the points $x_i = x_1 + \sigma(x_i - x_1), i = 2, n+1$; // Shrink
            Compute $f_i = f(v_i)$ for $i = 2, n+1$;
        end
    end
    else
        $x_c \leftarrow x(-\gamma, n+1)$; // Inside contraction
        $f_c \leftarrow f(x_c)$;
        if $f_c < f_{n+1}$ **then**
            Accept x_c ;
        else
            Compute the points $x_i = x_1 + \sigma(x_i - x_1), i = 2, n+1$; // Shrink
            Compute $f_i = f(v_i)$ for $i = 2, n+1$;
        end
    end
    Sort the vertices of S with increasing function values ;
end

Algorithm 3: Nelder-Mead Algorithm

Input: Inputs a, b, function f
Output: Root b or s
Calculate $f(a)$;
Calculate $f(b)$;
if $f(a)f(b) \geq 0$ **then**
  Exit function because the root is not bracketed;
end
if $|f(a)| < |f(b)|$ **then**
  Swap (a, b);
end
$c \leftarrow a$;
Set $mflag$;
repeat
  if $f(a) \neq f(c)$ *and* $f(b) \neq f(c)$ **then**
    $s \leftarrow \frac{af(b)f(c)}{(f(a)-f(b))(f(a)-f(c))} + \frac{bf(a)f(c)}{(f(b)-f(a))(f(b)-f(c))} + \frac{cf(a)f(b)}{(f(c)-f(a))(f(c)-f(b))}$;
  else
    $s \leftarrow b - f(b)\frac{b-a}{f(b)-f(a)}$; `// Secant method`
  end
  if s *is not between* $(3a + b)/4$ *and* b *or*
  (mflag is set and $|s - b| \geq |b - c|/2$*) or*
  (mflag is cleared and $|s - b| \geq |c - d|/2$*) or*
  (mflag is set and $|b - c| < \delta$*) or*
  (mflag is cleared and $|c - d| < \delta$*)* **then**
    $s \leftarrow \frac{a+b}{2}$; `// Bisection method`
    Set $mflag$;
  else
    Clear $mflag$;
  end
  Calculate $f(s)$;
  $d \leftarrow c$;
  $c \leftarrow b$;
  if $f(a)f(s) < 0$ **then**
    $b \leftarrow s$;
  else
    $a \leftarrow s$;
  end
  if $|f(a)| < |f(b)|$ **then**
    Swap (a, b) ;
  end
until *convergence*;
Output b or s (return the root);

Algorithm 4: Brent's Method (brentq)

Master's Theses in Mathematical Sciences 2024:E22
ISSN 1404-6342

LUTFMA-3534-2024

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/