Bachelor Thesis

Representing Market Data in Latent Space using a Transformer VAE

Author: Marcél Busschers (2640334)

GitHub

Supervisor: Peter Bloem

2nd Reader: Benno Kruit

July 12, 2021

Abstract

Autoencoders were created to allow for compressing, and encoding unlabeled data; for the sole purpose of reconstructing the data as close as possible. Its counterpart - the Variational Autoencoder - is a latent variable model, created to allow for generation of new data influenced by its latent space. The downside of Variational Autoencoders is that it was designed for image reconstruction, making it unsuitable for sequential data - such as Natural Language Processing (NLP). The pinnacle revelation of NLP was introduced as the Transformer model; a model that introduced attention as a process to weigh the relation of sequential data. In this paper, a Variational Autoencoder will be created by making use of Transformer blocks to allow for a more accurate representation of sequential data; this model is then applied to a different set of sequential data, financial (market) data, in order to show that the Latent Representation of the Variational Autoencoder could give insight to the financial market. In turn, helping analyse patterns that was previously missed by human analysis.

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

1 Introduction

The Variational Autoencoder (VAE), first introduced by Diederik P Kingma and Max Welling in 2013[14], is a latent variable model [16] that allows for generation of new data by compressing raw data into a latent representation. A VAE has many practical uses, but it is generally used for image generation or reconstruction [23]. That being said, a Vanilla VAE struggles with sequential data, and hence needs to be adjusted or modified with another model. This makes the VAE model flexible in what you can achieve with it. Samuel Bowman, and others, combined the use of LSTM models with a VAE architecture [4] for the purpose of sentence generation; making it suitable for Natural Language Processing (NLP).

There has been an improvement on NLP however; the introduction of the Transformer model by Vaswani [28] introduces attention to allow for a more accurate relation of sequential data. The Transformer model has opened a pathway to many successful models including BERT and GPT-2 [9, 22]. With Transformer blocks, one can not only create a model suitable for Language, but also for any data that's in sequence; hence, this paper is about the combination of the VAE and Transformer blocks to allow for a more accurate representation on sequential data in Latent Space.

In order to achieve a proper representation of the latent space, the creation of a VAE using Transformer blocks will first be explained. Furthermore, the creation of the model will initially be done to generate sentences in continuous space; after which, the model will be applied to a different set of sequential data - financial market data. By applying the model to the financial market, it can then be used to analyse the market by representing the raw data in latent space; in turn, showing general patterns and further analysis - this leads to the following research question:

Can any meaningful market analysis be made from visualising the Latent Space of a Transformer VAE?

To tackle such a research question, the model must be used to visualise the latent representation in two dimensional space for the entirety of the market data history. After which, it could be seen where a certain section of the market lies in projection of the latent space. By doing so, multiple sequences of the market must be analysed and compared, making this a quantitative research.

By conducting such a process, it was hypothesised that the latent space will represent certain distinct patterns in the market. Allowing for either a reassurance of manual analysis on the market, or bringing a certain pattern - in the market - to ones attention. It is also hypothesised that further predictions of the market outcome can be made by analysing the latent space of the financial sequences generated by the model.

Before the model can be discussed, a short section delving into the related work must be covered, as well as a little background to what has been done before. The Variational Autoencoder is something that has been well developed in the past, for which used techniques will be referred to. However, the use of the Transformer blocks were not altered from the original creation of Vaswani, therefore, no techniques of which will be referred to. In addition to the separate models themselves, further similar work of the entire model will be referred to. Additionally, a more in-depth explanation of the VAE and Transformer models will follow, and finally, a full explanation of the created model for this research will be covered.

2 Related Work

Samuel Bowman, and others, introduce the study of an RNN-based Variational Autoencoder that incorporates distributed latent representations of sentences [4]. This allows for the model to get a better syntactic representation of sentences; in turn, producing well-informed sentences through deterministic decoding. My research, however being similar, will be featuring a Transformer-based Variational Autoencoder whos syntactic representation can be applied to different sequential data.

In contrast to Samuel Bowman's architecture of using a RNN-based VAE, Stanislau Semeniuta, and others, propose a hybrid architecture consisting of fully feed-foward convolutional and deconvolutional components with a recurrent langauge model [24]; this in turn, allows for faster convergence and the ability to handle longer sequences. This model gets compared to a Bowman's LSTM VAE showing the outperformance of longer sequences.

As will be discussed in section 5.3, the VAE model used was known as a β-VAE. Harshvardhan Sikka explores the solution of entangled latent representations of object recognition by making use of a Beta-VAE [25]. This paper explores how similar observable variables can result in the same latent variables despite being different; This results in an entangled latent space, for which needs to be corrected, or disentangled, by making use of a Beta-VAE.

Section 5 covers experimentation done to solve posterior collapse. Solving posterior collapse is a large part of this research, for which James Lucas covers in why posterior collapse occurs in latent variable modelling [19]. Bohan Li, and others, propose a surprisingly effective fix for latent variable modelling of text [15]; by taking two known heuristics and applying them to achieve a surprising result that using ELBO as the `typical surrogate` objective for VAEs may not be sufficient for balancing the latent representation and data distribution.

3 Background

The custom created model stems off of two prerequisite models: the generic Variational Autoencoder, and Vawasni's Transformer model. This section will briefly go over the background of these two models. For a more in depth understanding of these two models, refer to Appendix B.

3.1 Variational Autoencoder (VAE)

This subsection covers the basic understanding of a Variational Autoencoder [21]; For a full explanation on VAEs, refer to Appendix B.1.

3.1.1 Architecture

The VAE consists of two separate models: The Encoder and Decoder. The Encoder is responsible for encoding raw data in compressed form; Whilst the Decoder is to reconstruct the compressed encoded form. The Encoder provides a probabilistic description of the observed variable in latent space. Therefore, the Encoder doesn't output a single value describing each latent state variable, but rather outputs a probability distribution for each latent variable. The decoder is a type of likelihood-generative model, for which the posterior of the model is approximated via variational inference. In Figure 1, it is shown that the Encoder outputs a mean and co-variance vector, this is used

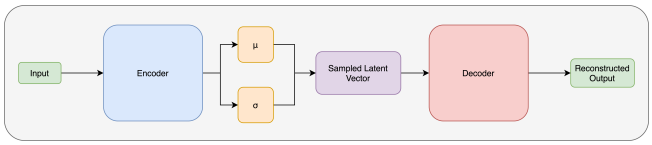


Figure 1: VAE Architecture

to sample a latent representation z via the reparameterization trick [11]. This latent representation is then fed through the Decoder to output a reconstructed input $\hat{x}$. To best understand later experiments, it is important to understand the loss function of this model: The Evidence Lower Bound (ELBO).

3.1.2 ELBO

The concept of this loss function is seeking to maximise the log-likelihood. It is composed of the encoder network $q_v(z|x)$ and the decoder network $p_w(x|z)$ as well as the standard normal distribution $p_w(z)$ being the marginal probability density on z given p.

$$loss(v, w) = KL(q_v(z|x), p_w(z)) - \mathbb{E}_q ln p_w(x|z)$$

Exploring the loss function, It consists of the KL divergence constructed between the Multivariate Normal Distribution that the encoder produces for x and the standard normal multivariate distribution; as well as, and the expectation under q of the log probability of x given z. This expectation is not easy to work out, making it difficult to back-propagate through, hence it gets approximated. This is the approximation of the posterior mentioned before.

Essentially, the loss function is composed of the summation between the Encoder's KL loss, and the Decoder's Reconstruction loss: $loss = KL + Reconstruction$. In a further section, it will be shown that approximating the posterior can lead to problems with reconstruction.

3.2 Transformer

This subsection covers the basic understanding of Vaswani's Transformer Blocks from the "Attention is all you need" paper [28]. For a full explanation on Transformer models, refer to the Appendix B.2.

3.2.1 Architecture

A Transformer block is composed of two layers: A Multi-headed-Attention layer followed by a Feed Forward layer. Multi-headed-attention is the operation of giving meaning from values to other values in a sequence by applying use of Self-Attention. Essentially, Self-Attention is the process of associating each individual instance in the sequence to every other instance in the sequence; this is achieved through the dot-product step seen in Figure 2. The input is matrix-multiplied by itself to produce a `score` matrix which determines how much focus should an instance be put on other instances. The dot-product outputs values between negative and positive infinity; therefore, a softmax operation needs to be applied to map the values between 0 and 1. The score matrix is then matrix-multiplied by the original input vector to get the output vector; doing so will multiply the softmax score by the input, creating a vector for which instances that are more important get a higher score than those which are less important. This is essentially how the Transformer model gives input sequences meaning.

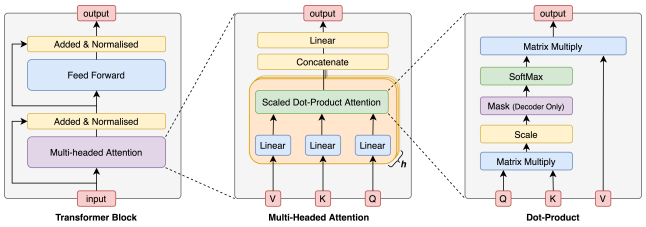


Figure 2: Transformer Block Architecture

3.2.2 Multi-headed Attention

The Self-Attention's dot-product can be repeated n times; each repetition, in theory, will learn something different and thus give the model more representation power. In order to

achieve this, the input is split into n equal vectors - also known as `heads`; they then each go through the same Self-Attention process, followed by a concatenation of each dot-product. This can be visualised in the Multi-headed-attention step of Figure 2.

The Multi-headed-attention output vector is added to the original input via a Residual connection which is then passed through a normalisation layer to produce point A; after which is fed through a point-wise feed-forward network (Linear layers with ReLU activation) for further processing. The final output is again added to point A via a Residual connection which is further passed through another normalisation layer to produce point B - the Transformer Block's output.

The point of these residual connections is to help the network train by allowing gradients to flow through the networks directly. The layered normalisations are used to stabilise the network which results in reducing the training time significantly. Finally, the point-wise feed-forward layers are used to further process the attention output, in turn giving it a richer representation. For a more in-depth, informal, explanation, refer to the blog post of Peter Bloem [3] or Jay Alammar [1].

4 Model Architecture

With an understanding of Variational Autoencoders and Transformer Blocks, the hybrid Transformer VAE will be discussed in this section.

4.1 Architecture

The Transformer VAE is constructed via an Encoder with n stacked Unmasked Transformer Blocks, and a Decoder with n stacked Masked Transformer Blocks. The output of the Encoder goes through the reparameterization trick [11] to properly sample the latent vector z; for which it get added to the input of each Transformer block in the Decoder. The output of the Decoder is a probability distribution representing the most likely form of reconstruction influenced by z. Figure 3, below, is a visual representation of the architecture; for a more in-depth visualisation, refer to Figure A.3.

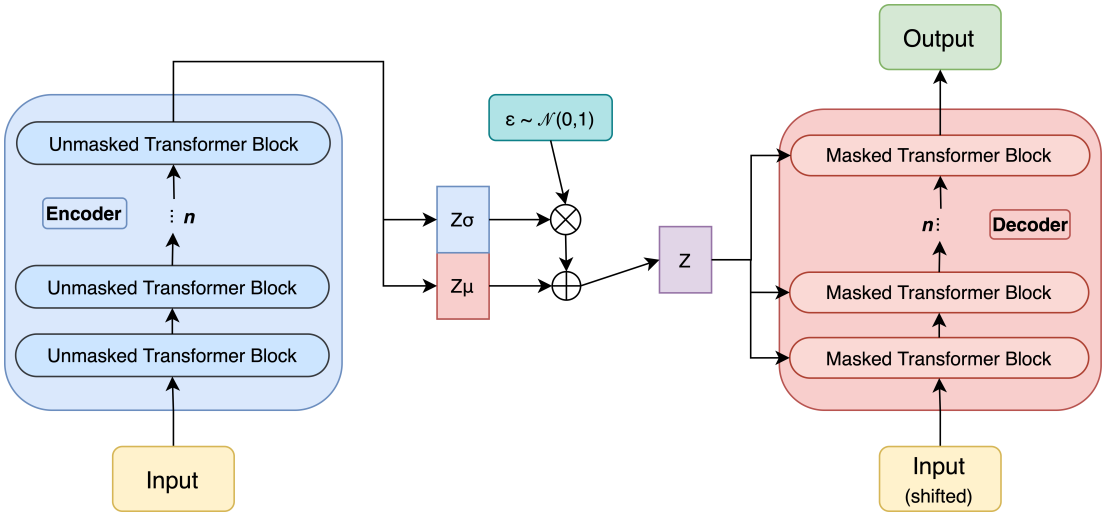


Figure 3: Transformer VAE Architecture

It is important to note that the Transformer blocks of the Decoder are masked, whereas those of the Encoder are unmasked. Masking is a known operation on sequence-to-sequence modelling [26].

Masking is essentially matrix-multiplying a given matrix by $-\infty$ to disallow the model to see any future instances; figure 4 visualises a matrix being masked with a red triangular matrix. This is an optional step taken during the dot-product stage of the Transformer block, where the `score` matrix is optionally masked to disallow the Decoder from seeing into the future.

The Encoder is needed to represent the data in latent representation so it can then train the Decoder; in doing this, it needs access to all future instances at a given point in time. However, the Decoder is responsible of reconstructing an instance at a given point in time; therefore, it is important that it may not see into the future since it has to learn how to predict what comes next; hence, the need for masked Transformer blocks.

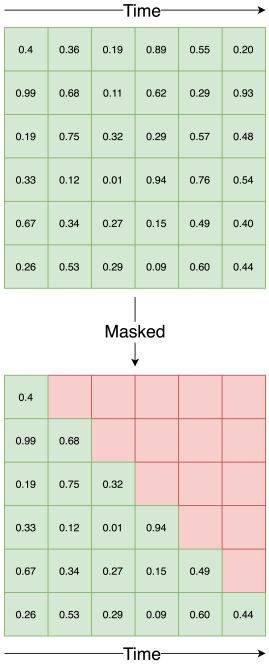


Figure 4: Masked Matrix

Finally, the input to the Decoder is the same as the input to the Encoder, shifted to the right by one instance. Ultimately, this chops off this last instance because it is the Decoders job to predict the last instance in the sequence. The input of each Transformer block is summed with the latent vector z in order to influence the model with the latent representation.

4.2 Data

The Transformer VAE is an unsupervised model, therefore no labeled data is required. Any sequential data can be used to train this model, so long as it is correctly loaded in as Torch Tensors, and the output distribution is altered to fit the type of data used. During this research, two sets of data was used: Language data, and Financial Market Data.

4.2.1 Language Data

Since Transformer models are widely used for Natural Language Processing, we first optimise the model using language data. In specific, a set of simple sentences averaging in 120 characters in length; this gets taken from the Coco dataset publicly available [8]. The Coco dataset is used for object recognition training; it consists of thousands of images with context - precisely 5 short sentences describing the image. In order to train the model on these sentences, we strip the data of all images leaving only thousands of simple sentences. The raw data can be found in the *GitHub*.

We load in the language data character by character; meaning the model would learn to give meaning between each character, not between each word. In doing so, the output distribution of the Decoder is softmaxed to give a categorial distribution over all characters for each point time time, seeing the model needs to give the most likely character to come next in the sequence. This is achieved using a *Log Softmax* function on the output of the Decoder. The Decoder's Maximum Likihood loss function for the reconstruction is then a *Negative Log Likelihood*.

4.2.2 Financial Data

In order to answer the research question, the model needs to be applied to a different situation of financial data. For this research, we use historical Forex financial data [18]; in specific, over 13 years worth of EURUSD data on a 4 hour time period. This data consists of OHLC data [20],

meaning that it is two-dimensional. The model is designed to operate using one dimensional sequences; therefore, to reduce the OHLC data down to line data, we take the `close` column only - giving a one-dimensional sequence of market data.

In a financial timeseries, the distribution is no longer over all instances for each point in time, but rather a 1D real-valued space for *every* point in time. Therefore, the distribution becomes a 1D Gaussian. Thus, since a Gaussian output refers to a `mean` and `co-variance` vector, we set the Decoder's Maximum Likelihood loss function to a custom *Gaussian Negative Log Likelihood* loss shown by the following formula:

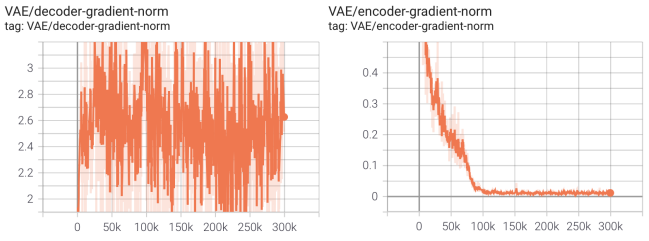
$$-\frac{1}{2}\sum_{i} ln\sigma_i + \frac{1}{\sigma_i^2}(x_i - \mu_i)^2$$

4.3 Training

With a set data source, we then train the Transformer VAE to establish a base-run using the language data. In order to evaluate the performance of the model, we plot the Gradient Norms separately for the Encoder and Decoder, as well as the KL and Reconstruction loss curves.

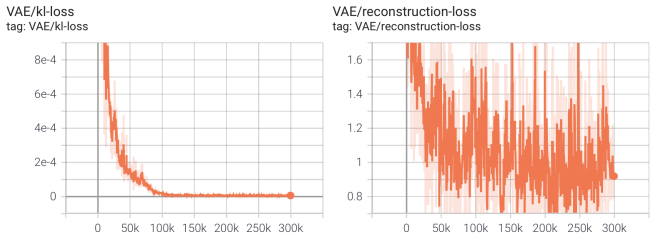
For an explanation of each hyperparameter, refer to Table 2 in Appendix A. Training the model for 150+ epochs, on a depth of 12 resulted in a known issue in VAEs: posterior collapse [19], or otherwise known as Decoder collapse.

In theory, a perfectly trained VAE, without Decoder collapse, occurs when the Encoder produces stable `mean` μ and `co-variance` σ vectors; in turn, sampling a good latent representation z. In other words, when the Encoder distils useful information from its input to μ and σ.



(a) Gradient Norms

As shown in Figure 5(a), the Encoder experiences a stable back-propagation as the gradient makes its way down to 0. However, the Decoder experiences a highly unstable back-propagation, meaning to say it is not learning anything useful; this can be seen in the reconstruction loss curve of Figure 5(b).



(b) Loss Curves

Figure 5: Transformer VAE Base-Run

Example Reconstruction An input sentence "*A person riding a horse in a horse park while another horse wonders in nearby grass.*" reconstructs into an ungrammatical sentence of "*A woman with a mounter a table with a bat and chicken flying kite.*".

The posterior collapses when the Encoder fails to distil useful information from its input x to parameters μ and σ, leading to a signal that is either too `weak` or too `noisy`, thus creating a entangled latent space; as a result, the Decoder becomes too powerful of a model for the Encoder's latent representation, in turn, the Decoder starts ignoring z samples produced by the Encoder.

5 Experiments

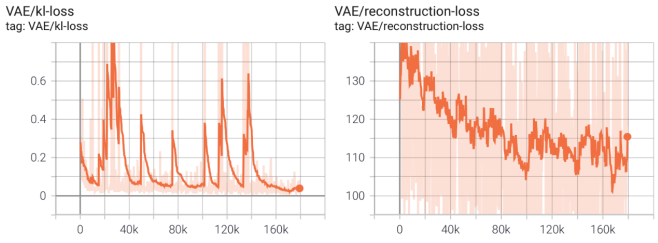
We sought out a variety of techniques in order to solve this Decoder collapse; three of which will be explored in this section. The experiments were conducted on the language data, seeing as the model should first be optimised for text generation before applying it to the financial dataset.

5.1 Word Dropout

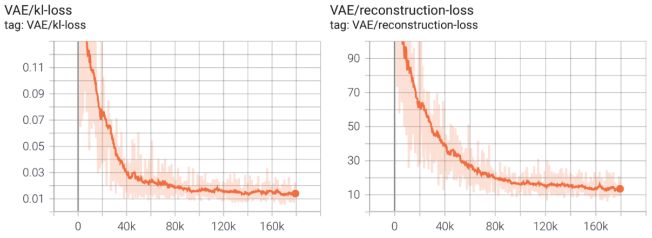
The first experiment involves weakening the Decoder of the VAE to avoid posterior collapse. As mentioned in `section 4.3`, the posterior collapses when the signal from input x to parameters μ and σ are too weak; this means the Decoder is too powerful of a model for the Encoder. Therefore, one could attempt to weaken the Decoder to avoid posterior collapse by introducing word dropout in the Decoder [4].

Word dropout involves dropping a random set of instances from the Decoder's output; this is achieved by tuning a probability hyperparamter indicating how many random instances to drop. Teng Long et al shows that applying a heavy word dropout leads to a non-zero KL loss, however, also to a worse log-likelihood [17]; this indicates that applying word dropout is not sufficient in solving posterior collapse.

Figure 6(a) shows the model being training on a word dropout of 50%; the plots are intensely smoothed to see the direction of the loss curves, however, 50% dropout leads to a noisy reconstruction with very little being learnt. Increasing the dropout rate to 60%, figure 6(b) shows a massive improvement. However, the reconstruction loss flattens around the value 15, which proves that applying word dropout is not sufficient in solving decoder collapse.



(a) 50% Dropout



(b) 60% Dropout

Figure 6: Word Dropout Results

Bowman shows that when increasing the dropout rate, the amount of information stored in the latent variable increases [4, p. 7]; hence, the Decoder is influenced more by the Encoder's representation, leading to better reconstruction. However, increasing the dropout rate too high leads to ungrammatical English sentences. Generated sentences for run `b` can be found on the *GitHub*.

5.2 Free-bits

Diederik Kingma et al introduces another objective to reach a better optima: a constraint on the minimum amount of information per group of latent variables - free-bits [13, p. 14–15].

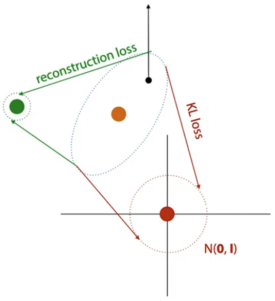


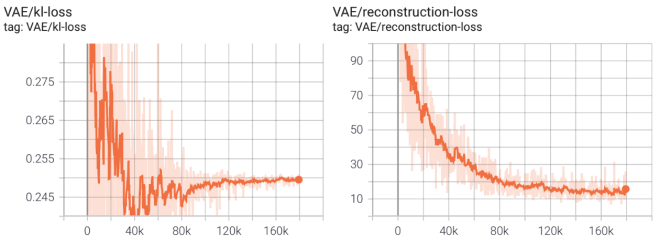
Figure 7: Loss Visual [2, s. 88]

A regular ELBO loss can be visualised in figure 7, where the KL term is pulling the loss into Normal form, whereas the NLL term is trying to achieve the best reconstruction possible; in turn, the loss gets pulled into a Multivariate Normal Distribution.

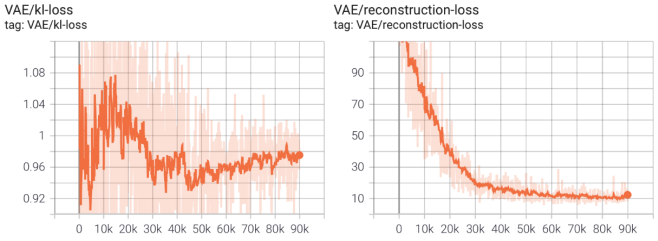
If the strength of the KL term were to be limited, the loss would shift more to better reconstruction. In theory, this is what free-bits does; by adding a constraint to the KL term, lambda λ, it prevents the KL loss from getting too low.

$$D_{KL} = maximum(\lambda, D_{KL})$$

In figure 8(a), the results are shown of training the model with a lambda constraint of 0.25 - in other words, preventing the KL loss term from going below 0.25. This again, is an improvement from the base run as the reconstruction loss is quite stable; however, still averages off around the value 15. In an attempt to limit the KL term even more, figure 8(b) shows the training of the model with a lambda constraint of 1.0; the reconstruction loss here makes its way further down to an average of 10, with surprisingly good reconstructions.



(a) 0.25 Lambda



(b) 1.0 Lambda

Figure 8: Free-bits Results

Example reconstruction We reconstruct the sentence "*A man looking at a picture of a TV screen.*" to "*A woman laying in a hospital bed hooked up to a TV monitor.*". Here, a lot of similarities in the two sentences can be seen; for instance, the model could be seeing a similarity in the two words `man` and `woman`, as well as, `screen` and `monitor`.

5.3 Beta Annealing

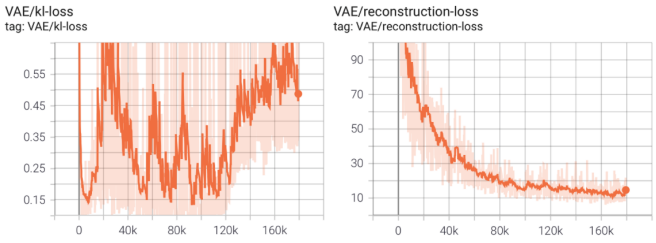
The final experiment is widely common in the area of VAEs - Beta Annealing [6, 12]. Beta Annealing consists of clipping the KL loss with a Lagrangian multiplier β.

$$D_{KL} = \beta \cdot D_{KL}$$

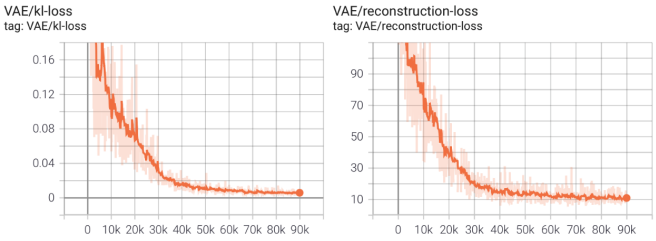
In section 4.3, it was mentioned that the posterior collapses when the signal from input x to paramters μ and σ is either too weak or too noisy. The previous two experiments was in accordance with a `weak` signal. To combat a `noisy` signal, a `Beta-VAE` needs to be implemented.

When $\beta = 1$ it is a regular VAE. However, clipping the β between 0 and 1, it applies a weaker constraint on the latent representation and enhances the representation capacity of z.

Figure 9(a) shows the model being training with a beta value of 0.5. The KL Loss curve starts increasing at a point, which is a good sign in VAEs; it essentially means that the Encoder is starting to work harder in creating more meaningful latent representations for the Decoder to understand, however, it also implies that the latent space is becoming more entangled.



(a) $\beta = 0.5$



(b) $\beta = 2.0$

Figure 9: Beta-VAE Results

When applying a $\beta > 1$, it applies a stronger constraint on the latent representation and therefore limits the representation capacity of z; in turn, keeping a disentangled latent space [29]. Therefore, there is a trade-off between solving a `noisy latent space` and solving `reconstruction accuracy`. We show, via figure 9(b), an example run with a beta value of 2; the KL term shows the encoder is no longer working as hard to distil useful information, however, the reconstruction is better - which can be shown via an example reconstruction.

Example reconstruction We reconstruct the sentence "*A woman stands beside a large brown horse.*" to "*A woman sitting next to a large statue on a bench.*". Here, many words that were correctly reconstructed can be seen - `A woman` and `large` in this case. The model also seemed to find a similarity between `stands` and `sitting`, as well as, `beside a` and `next to a`.

6 Method

With several methods of an optimised model, we could then address the research question. *Can any meaningful market analysis be made from visualising the Latent Space of a Transformer VAE?* In order to address this question, we had to adjust the model to the financial market data, which then means adjusting the output distribution to a Gaussian distribution. Furthermore, in this research question, there is direct attention to the latent space; therefore, it is more important to focus on a disentangled latent space than for best reconstruction. It is not entirely important for the reconstruction to be perfect, so long as it retains important data about the market - such as peaks, trend lines, etc.

For the remainder of this research, we used a Beta-VAE as the primary model - explained in section 5.3. Due to the need of focusing attention on the latent space, it was necessary to set the hyperparameter, beta, to be greater than 1 - specifically $\beta = 2.0$. In doing so, we prioritise a disentangled latent space over the reconstruction quality. Due to time constraints, the model was trained using a depth of 2, as this allows the model to converge quicker when running multiple tests.

Firstly, we needed good reconstruction, so the model was trained on the financial market data while tuning the hyperparameters to attain a good reconstruction. To achieve this, we first start training with single batching without any compression on the latent size, while adjusting the number of epochs until a good reconstruction is attained.

With good reconstruction, we compress the data by altering the latent size to 25% of the full sequence length; again training until reconstruction was good. Finally, single batching could be turned off, allowing the model to get good reconstruction on all the data.

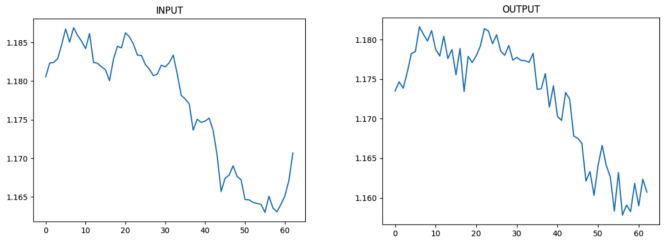


Figure 10: Final Reconstruction - Market data

Figure 10 shows the final reconstruction the model achieved on the financial data. The output is slightly overfitted, however, the important information of the market was kept; such as the overall shape, trend, and peaks. Once the model was fully trained, we could generate the latent space; however, since the latent representation is embedded in high dimension, the embedding dimensions need to be lowered to 2D in order to plot. In doing this manually, the distance between instances becomes inaccurate; therefore, to plot an accurate representation of the high-dimensional latent space in 2D, we use SciKit Learn's [5] *t-SNE* [27, 33] - a tool that allows you to visualise high-dimensional data.

With a plotted latent space, hundreds of different financial sequences were then passed through the Encoder to see where on the latent space they lie. At this point, the research is purely quantitative, meaning hundreds of sequences had to be analysed in order to pick up on meaningful patterns that could be seen within the latent space.

7 Results

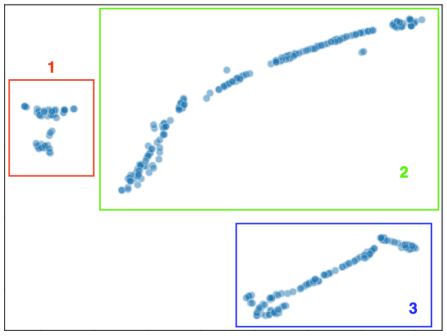


Figure 11: Latent Space

Figure 11 shows the latent space across all the financial data, for which we can see 3 distinct clusters. Cluster 1 is small, and dense, which could indicate a distinct class of sequences; whereas cluster 2 is is elongated, which could indicate a series of similar classes. As for cluster 3, it is far away in distance from cluster 1 and 2; this could indicated that the class is either very different from the other two, or the sequences within are varied or unstable.

Upon manual analysis of hundreds of sequences, distinctions could be made about the 3 clusters. Cluster 1 seemed to characterise with a lot of sudden peaks, which is unique within this cluster. Three examples of which can be visualised in figure A.4 in Appendix A. Cluster 2 would focus more on trendlines, which is a broad classification; trendlines come in a variety of different forms, two major distinctions being up-trends and down-trends. Three examples of which can be visualised in figure A.5. As for cluster 3, the sequences varied quite a lot as it features a lot of unstable sequences. A lot of peaks and valleys were perceived in cluster 3, as well as reversals in trends. Three examples of which can be visualised in figure A.6.

`Cluster 1`	`Cluster 2`	`Cluster 3`
Sudden Peaks - Fundamental News Trading	Trendlines - Up & Down	Unstable market - Valleys & Peaks

Table 1: Major Cluster Distinctions

We can further analyse each cluster, producing further separation. Figure 12 shows the separation of each cluster into minor groups. An observation between analysing within the second cluster is the type of trendline that occurs within the latent space between minor clusters. For example, in figure 13, a downtrend taken from cluster 2.1 in subplot `a` shows a consistent, stable, downtrend; whereas the downtrend taken from cluster 2.4 in subplot `b` shows an more ragged, unstable, downtrend. Therefore, certain market patterns can be identified depending on where they lie in latent space.

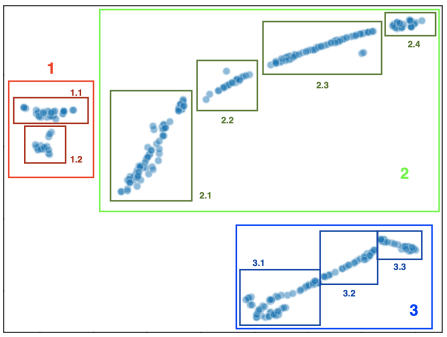
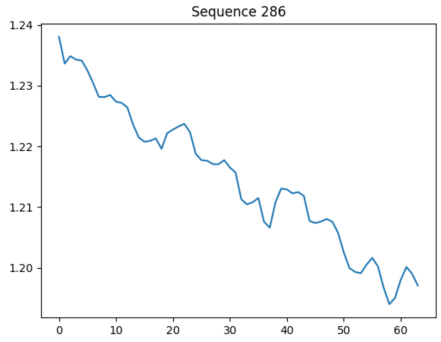
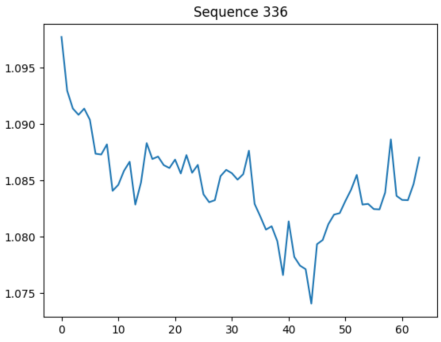


Figure 12: Latent Space Minor Groups



(a) Sequence from 2.1



(b) Sequence from 2.4

Figure 13: Sequences in Same Cluster

In further analysis of sequences, it was commonly found that the following sequence is quite often found in a neighbouring minor cluster; this in turn could assist one in making a prediction of the coming market. For instance, in figure 14, sequence 1 was identified in minor cluster 2.1 with a stable downtrend; following it, sequence 2 was identified in the neighbouring minor cluster 1.2 which shows a sudden peak within the market. With this logic, one could make assumptions about the future market allowing you to assume that, for instance, this downtrend, classified in sequence 1 of figure 14, might lead to the market performing a sudden rise and decline of price. One could combine this with potential fundamental news; for instance, if you are aware that the latest Nonfarm Payroll report [7] is due in the near future, you could assume that the market will accommodate the attributes of cluster 1.

It has also been observed that sequences within minor cluster 3.1 tend to follow with more sequences within that cluster. An example being every consecutive sequence from sequence 7 until 18 were all within minor cluster 3.1.

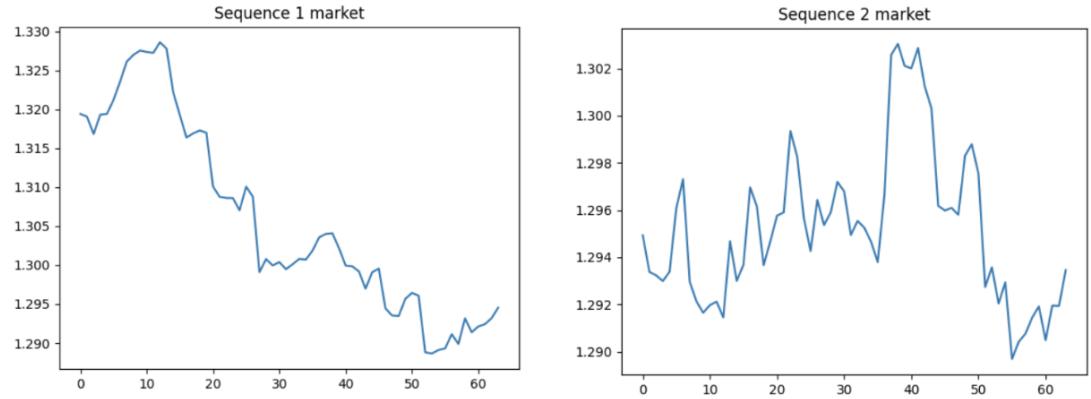


Figure 14: Analysis to Prediction

In addition to the observation of neighboring minor clusters, two minor clusters in specific seem to have high correlation with each other - 1.2 and 2.1. It is common to see sequences from 2.1 interpolate into 1.2 and vice versa. Meaning, it is common to see markets with sudden peaks suddenly enter a trend, as well as, trends into sudden peaks.

Since this is manual analysis of quantitative data, we could technically observe more patterns; however, the overall effect of this research has been seen. To see all the sequences with attention to their respective place in the latent space, as well as those grouped for the major and minor clusters, refer to the *GitHub*.

8 Conclusion

The creation of the Transformer VAE being applied to a different scenario shows just how flexible the model can be. This model, in theory and in practice, can be applied to multiple sets of sequential data for different analysis and prediction, such as weather prediction, sentence completion detection, or even genome sequencing.

It has been shown that VAEs experience a known issue of posterior collapse; for which the experimentation of word dropout, free-bits, and beta annealing was conducted to solve this collapse. It was also showed that it can be altered to focus more on disentangling the latent space than on perfect reconstructions. In doing so, the model can be optimised for any scenario it is used for; which, again, shows its flexibility. However, these solutions are, in-fact, quick-fixes and therefore don't always work as intended; they require hyperparameter tuning and testing to get a viable model.

Applying this custom model to financial data in order to answer the research question "*Can any meaningful market analysis be made from visualising the Latent Space of a Transformer VAE?*" was completely doable in this case. In doing so, it was possible to observe the latent space to make meaningful analysis. To answer the question, meaningful market analysis can, indeed, be made by analysing the latent space. However, at this point in time, the analysis made is quite limited to basic pattern recognition and simple observations made by manual analysis; this is due to a range of insufficient factors, such as a lack of data, not enough training, or too low of a depth.

Further manual analysis - or even Machine Learning practices - on the quantitative data would need to be done by professionals of the respective field of Market Analysis; doing so would result in more patterns being found as the help of a professional doesn't have a limited knowledge of the financial market. Therefore, this model is useful for analysis purposes, as an technical indicator; however, like every indicator, it should not be used for prediction sake, but rather as an indication or confirmation of human analysis.

As it stands, the model was trained on EURUSD data only; thus, only making meaningful analysis for that market alone. The model would need to be trained either separately on other markets, or a concatenation of all major markets to allow the model to learn a more *overall* approach; resulting in a more generalised latent space.

Furthermore, due to time constraints, the model had to be trained with a low `depth` to allow for recreation of Decoder collapse during the experimentation phase. Training with a higher depth, for longer periods of time, should result in a better latent representation, in turn, a better latent space and reconstruction overall. Further experimentation could also be made to test the latent space, such as interpolation of sentences.

That being said, it should also be noted that the posterior collapse was not fully solved, so it cannot be concluded that the model is as accurate as it could be. In addition, an inaccurate latent space is also plausible due to insufficient tSNE configurations, or hyperparameter tuning. Nevertheless, the model performed quite well and produced a quantitative set of analysable results, leading to a fully adaptable, and potential market indicator.

References

[1] Jay Alammar. The illustrated transformer, Jun 2018. URL: https://jalammar.github.io/illustrated-transformer/.

[2] Peter Bloem. 9.4 variational autoencoders - vrije universiteit amsterdam - machine learning. URL: https://mlvu.github.io.

[3] Peter Bloem. Transformers from scratch, Aug 2019. URL: http://peterbloem.nl/blog/transformers.

[4] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space, 2016. arXiv:1511.06349.

[5] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pages 108–122, 2013.

[6] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae, 2018. arXiv:1804.03599.

[7] James Chen. What are nonfarm payrolls?, Sep 2020. URL: https://www.investopedia.com/terms/n/nonfarmpayroll.asp.

[8] CVDF, Microsoft, Facebook, and Mighty Ai. Common objects in context. URL: https://cocodataset.org/.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. arXiv:1810.04805.

[10] Carl Doersch. Tutorial on variational autoencoders, 2021. arXiv:1606.05908.

[11] Gregory Gundersen. The reparameterization trick, Apr 2018. URL: https://gregorygundersen.com/blog/2018/04/29/reparameterization/.

[12] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. ICLR 2017 conference submission, 2016.

[13] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow, 2017. arXiv:1606.04934.

[14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014. arXiv:1312.6114.

[15] Bohan Li, Junxian He, Graham Neubig, Taylor Berg-Kirkpatrick, and Yiming Yang. A surprisingly effective fix for deep latent variable modeling of text. arXiv preprint arXiv:1909.00868, 2019.

[16] John C Loehlin. Latent variable models: An introduction to factor, path, and structural analysis. Lawrence Erlbaum Associates, Inc, 1987.

[17] Teng Long, Yanshuai Cao, and Jackie Chi Kit Cheung. On posterior collapse and encoder feature dispersion in sequence vaes, 2020. arXiv:1911.03976.

[18] Forex Software Ltd. URL: https://forexsb.com/historical-forex-data.

[19] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. Understanding posterior collapse in generative latent variable models, 2019.

[20] Cory Mitchell. Ohlc chart definition and uses, May 2021. URL: https://www.investopedia.com/terms/o/ohlcchart.asp.

[21] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. arXiv preprint arXiv:1609.08976, 2016.

[22] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

[23] Abhinav Sagar. Generate high resolution images with generative variational autoencoder, 2021. arXiv:2008.10399.

[24] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation, 2017. arXiv:1702.02390.

[25] Harshvardhan Sikka. A deeper look at the unsupervised learning of disentangled representations in β-vae from the perspective of core object recognition, 2020. arXiv:2005.07114.

[26] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mass: Masked sequence to sequence pre-training for language generation. arXiv preprint arXiv:1905.02450, 2019.

[27] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(11), 2008.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. CoRR, abs/1706.03762, 2017. URL: http://arxiv.org/abs/1706.03762, arXiv:1706.03762.

[29] Lilian Weng. From autoencoder to beta-vae, Aug 2018. URL: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html.

[30] Autoregressive model, May 2021. URL: https://en.wikipedia.org/wiki/Autoregressive_model.

[31] Bayesian inference, Jun 2021. URL: https://en.wikipedia.org/wiki/Bayesian_inference.

[32] Kullback–leibler divergence, Jun 2021. URL: https://en.wikipedia.org/wiki/Kullback\OT1\textendashLeibler_divergence.

[33] T-distributed stochastic neighbor embedding, Jun 2021. URL: https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding.

A Appendix

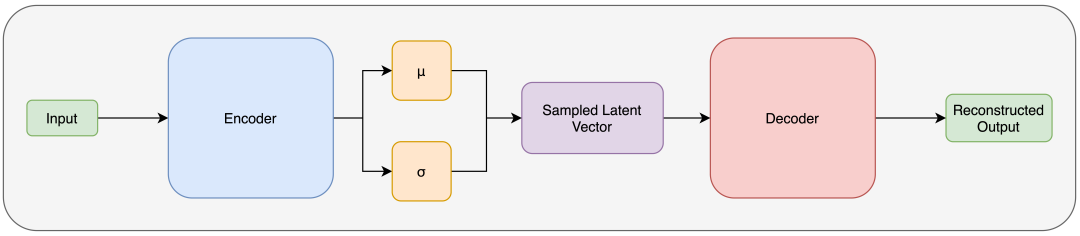


Figure A.1: Variational Autoencoder Architecture Enlarged

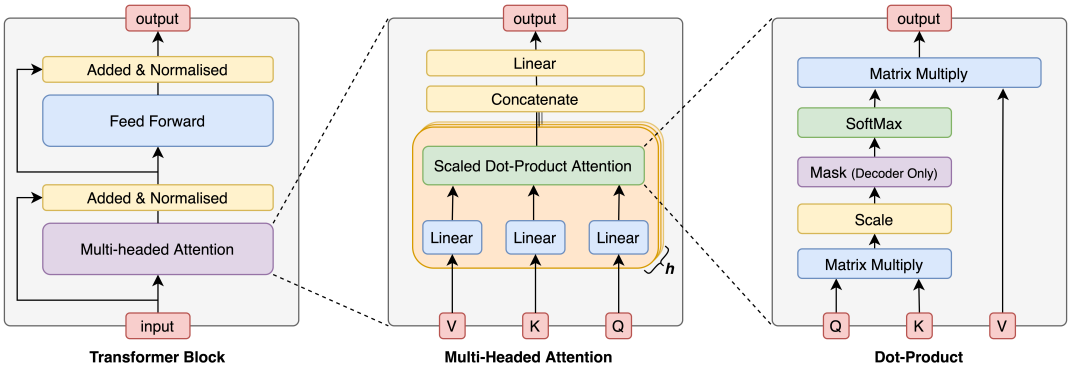


Figure A.2: Transformer Block Architecture Enlarged

Hyperparameter	**Description**
Number of Epochs	Spcifies how many iterations of the training loop
Batch Size	Specifies the total amount of instances of data to appear in a single batch
Learning Rate	The learning rate specified for the optimised - Adam
Embedding Size	The dimension size of the embedding vectors within the Transformer blocks
Heads	Number of attention heads to apply during the Multi-Headed attention step
Context	Length of the sequences extracted from the corpus (and the context used during inference)
Depth	Amount of stacked transformer blocks within the Encoder and Decoder
Log	Boolean attribute to log Tensorboards
Beta	Beta annealing character used to clip the KL loss
Message	A string used for manual entry of a description - gets saved to the generate.txt file
Dropout Probability	The probability value for Word dropout - done only on the Decoder
Free-bits	Sets a constraint on the KL loss
Latent Size	The amount of dimensions within the Latent Vector z
Single Batch	Boolean attribute for training on a single batch - to allow for overfitting

Table 2: Hyperparamters

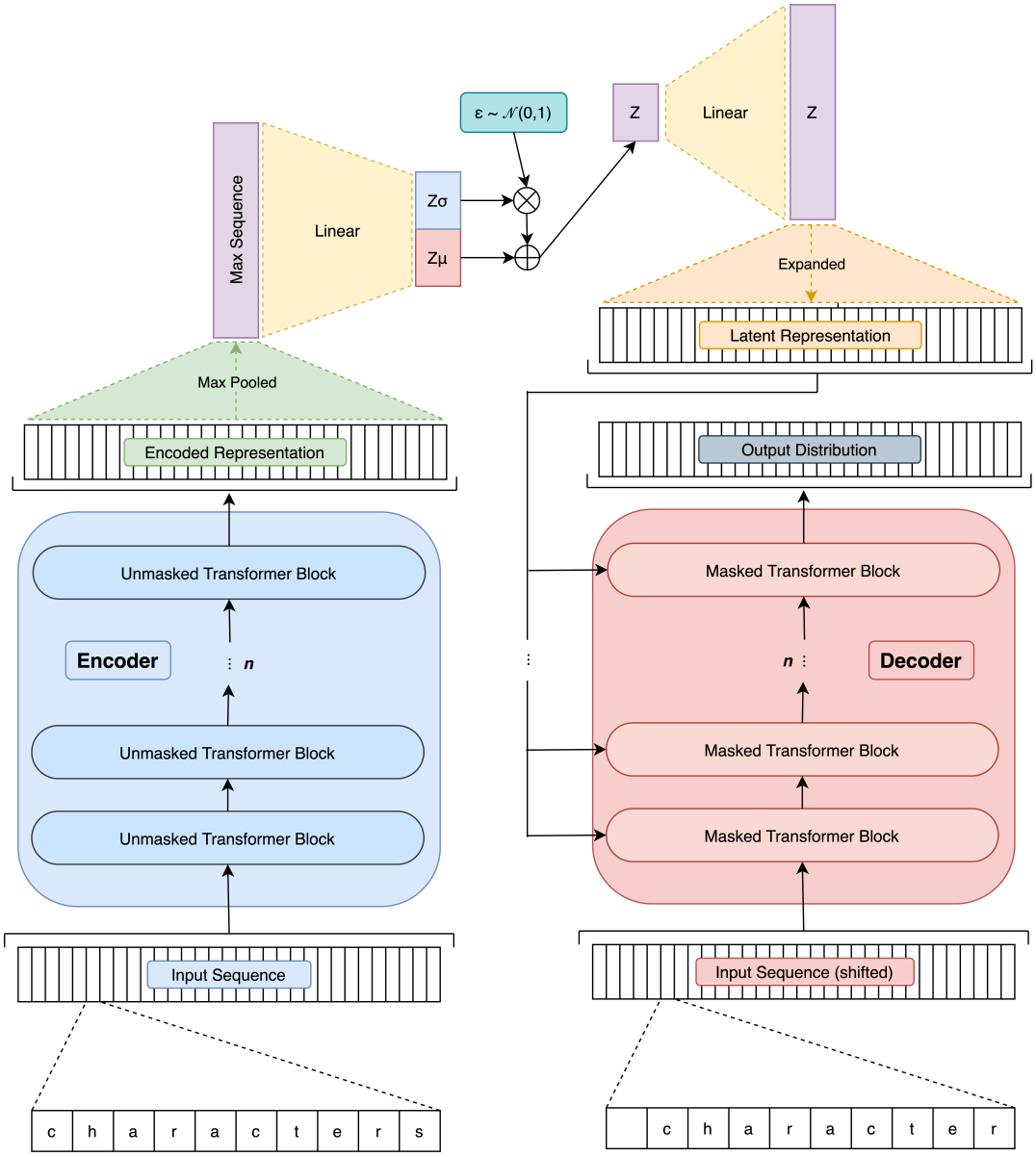


Figure A.3: Transformer VAE Architecture Detailed

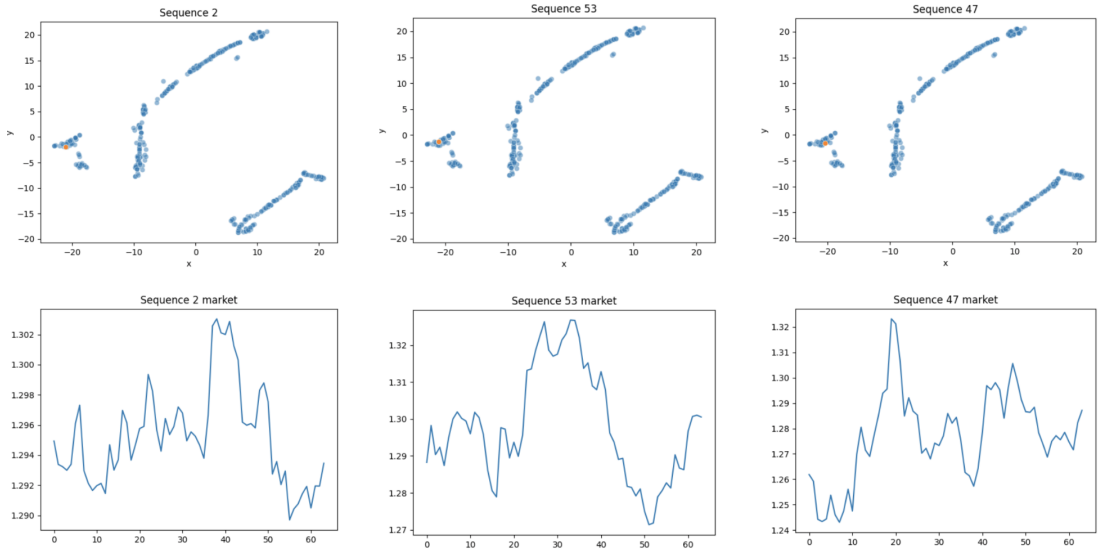


Figure A.4: Cluster 1 Results

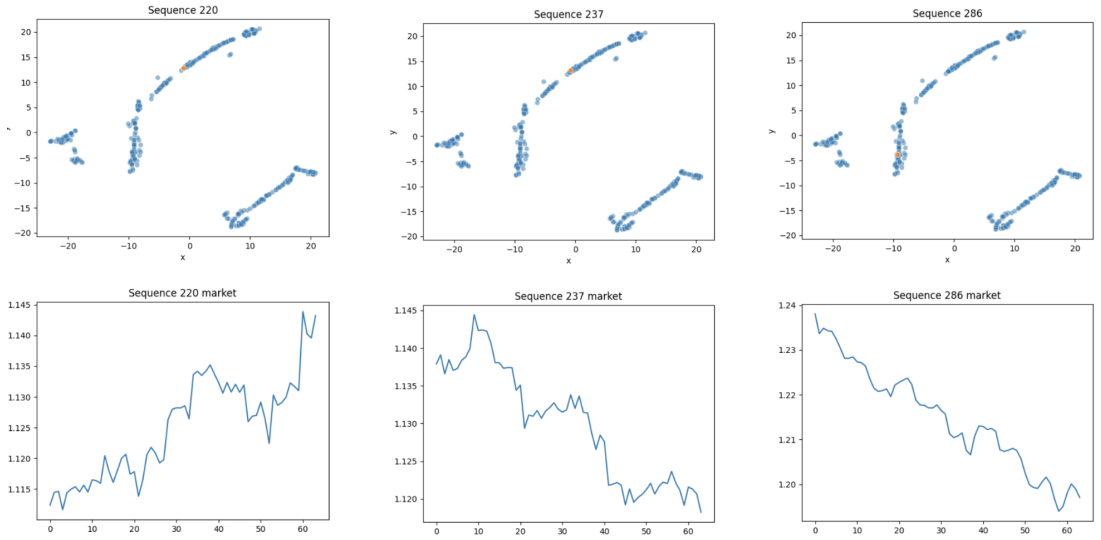


Figure A.5: Cluster 2 Results

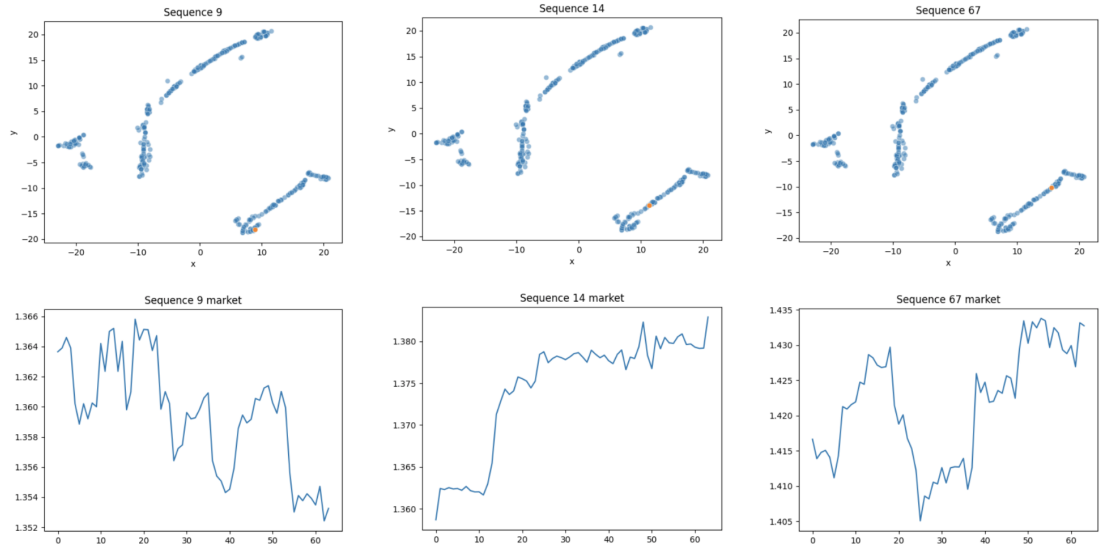


Figure A.6: Cluster 3 Results

B Appendix

This appendix is focused on a more in-depth tutorial of the two used models - the VAE and Transformer model - as a prerequisite. These explanations will, in no need, be fully explained; for a more visual explanation, the two blog posts by Peter Bloem[3] and Jay Alammar[1] could be worth reading.

B.1 Variational Auto Encoder (VAE) [2]

The VAE was first introduced by Diederik Kingma in 2014 [14]. For a more academic understanding, refer to the articles by Yunchen Pu et al, and Carl Doersch [10, 21].

B.1.1 What is it?

An Autoencoder is essentially a two-part model consisting of the Encoder and Decoder. The Encoder takes in raw data and converts it to an encoded representation; this representation is in compressed form - known as the Latent Space Representation. The job of the Encoder is to capture as much important information about the data as possible as it compresses it. The Decoder then takes this compressed representation, and learns to decompress it, and reconstruct it. The Latent space representation consists of Latent attributes for which the Encoder observed and chose to keep; the remaining attributes gets lost during compression - the hidden variables. The Decoder tries to generate the hidden variables, and reconstruct the observed variables.

The Variational Autoencder counterpart provides a probabilistic manner for describing an observation in latent space. Therefore, the Encoder of the VAE outputs a probability distribution for each latent attribute instead of outputting a single value describing each latent state attribute. Therefore, the model is enforcing a continuous latent space representation instead of a discrete one; in doing so, it becomes possible to sample randomly along these distributions, giving the possibility of generating completely new data.

B.1.2 Statistics

Suppose the decoder takes some hidden variable z and generates an observation x; attribute x can be seen, but it is unknown from which characteristics of z it came from. Therefore the probability of z given x needs to be computed. This can be done by using Bayes' rule[31]:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

However, computing $p(x) = \int p(x|z)p(z)dz$ is quite difficult, so $p(z|x)$ gets approximated by introducing another distribution $q(z|x)$. By defining parameters $q(z|x)$ to be similar to $p(z|x)$, they can be used to approximate the inference of the distribution.

B.1.3 KL Divergence

The KL Divergence is used to measure how different two probability distributions are to each other[32]. To ensure that $q(z|x)$ and $p(z|x)$ are similar, the KL divergence must be minimised:

$$min\, KL(q(z|x)||p(z|x))$$

Deriving the formula results in maximising the following:

$$\mathbb{E}_{q(z|x)}\, log\, p(x|z) - KL(q(z|x)||p(z))$$

Where the first term represents the likelihood of reconstruction and the second term insures that the learned distribution q is similar to the preceding distribution p. In this case, $q(z|x)$ represents the Encoder, whereas $p(x|z)$ represents the Decoder.

In conclusion, the model's loss function - The Evidence Lower Bound (ELBO) - consists of two terms: Minimising reconstruction error and maximising the similarity between the two distrubutions p and q.

$$ln\, p(x) = \mathbb{L}(x, \hat{x}) + KL(q(z|x)||p(z))$$

B.1.4 Sampling

The two terms that make up the loss function can be visualised in figure 7, where the KL term is pulling the the distrubution into Normal form, whereas the reconstruction term is trying to get the best reconstruction possible; as a result, the distribution gets pulled into a Multivariate Normal form.

The Encoder outputs a mean μ and co-variance σ vector to allow for sampling of the z vector. Sampling is done by taking a random point from the MVN distribution; however, doing so directly results in the inability to calculate the relationship of each parameter in the network by backpropagating. Instead, a random sample from a normal distribution ϵ is taken, shifted by the distribution's μ and scaled by the distribution's σ, in turn sampling z. This is known as the `reparameterization trick`[11]. By doing this, the model can then backpropagate through weights, allowing to effectively update accordingly.

$$z = \mu + \sigma \cdot \epsilon$$

B.1.5 Implementation

With a visual guide from figures A.1 and A.3, the model can be understood as a whole. Raw input data gets passed into the Encoder, for which it gets encoded into a compressed representation μ and σ; z get sampled via the reparameterization trick which then gets passed into the Decoder. Finally, the Decoder reconstructs the input via the information it gathers from z.

Once the model has a reconstruction, the loss can be calculated to see how far off the overall model is from representing the data to reconstructing it. With the loss calculated, the model can backpropagate to update the weights accordingly. Furthermore, the latent space z is nothing but a vector, and thus can be visualised by plotting it. Also, by adding another vector to it, you can interpolate data and manipulate it appropriately.

B.2 Transformer Model [3]

The Transformer Model was first introduced by Vaswani, et al, in 2017 [28]. For a more academic explanation, refer to the article by Jacob Devlin, et al, on the BERT model [9].

B.2.1 What is it?

The Transformer model, introduced by Vaswani, is a technological marvel in the NLP industry; it consists of an Encoder Decoder architectural network. The model is made up of `Transformer Blocks` which maps an input sequence to an abstract, continuous, representation - containing all the meaningful analysis of the input; This meaningful analysis consists of scores that represent how much meaning each instance has on every other instance within the sequence.

B.2.2 Input Embedding

When a input sequence gets passed into the model, a lookup table gets generated by making use of an Embedding layer; it fundamentally maps a categorical sequence to a vector of continuous values to accordingly represent it. This operation is known as the `Input Embedding` step.

The Transformer model has no recurrent information, meaning is has no information about the location of each instance within the sequence. To solve this, the input embedding gets added with `positional embedding` vectors. In doing so, information about where the instance lies gets added to the input, allowing the model to distinguish more accurately the word order of sentences.

B.2.3 Self-Attention

Fundamentally, the entire model revolves around the self-attention operation. Self-Attention is a sequence-to-sequence operation that outputs a score matrix of every instance mapped to every other instance. To best understand this, refer to figure B.1.

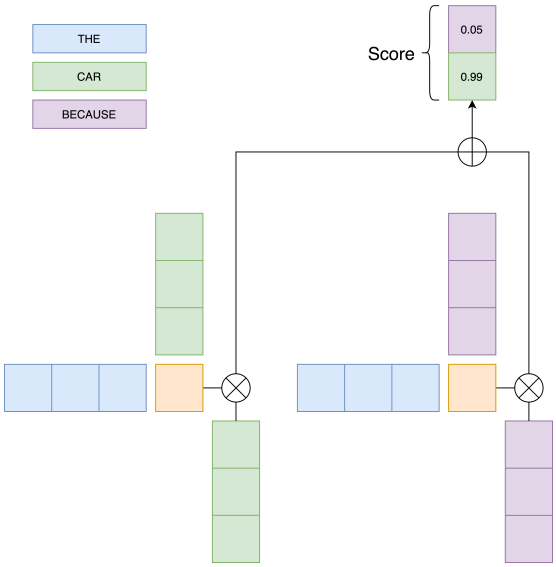


Figure B.1: Dot-Product Example

Three vectors representing three words: `The`, `because` and `car`. Applying the dot product on the words `because` and `car`, with respect to the word `the`, results in a score vector that applies to the relationship between `the` and the two words `because` and `car`. "***The car***" makes more grammatical sense than "***The because***", hence, the score for the word `car` is high, whereas the score for the word `because` is low.

This gets done by all vectors, creating a `score matrix` representing all instances mapped with every other instance (including itself); Irrelevant instances will have a low score, or even negative scores. It is important to note that this matrix is just a matrix, there is no ordering or sequential positioning involved. Computing the dot-product on a mixed sequence will result in the same scores for each instance, hence the positional embedding that gets done before the self-attention operation is highly important. Furthermore, these vectors representing the three words `the`, `because` and `car` gets created through the input embedding process mentioned in section B.2.2.

B.2.4 Text Generation Transformer

During this research, the type of Transformer model used was an autoregressive model [30]; where the output of the model depends on what comes before it. In other words, the output of the model becomes the input of the model for the next instance generation.

In order to use self-attention with an autoregressive model, the future of the sequence may not be seen by the model when predicting the next instance. This is known as `masking` and is explained section 4.1.