Generating Fashion Image Data with Convolutional Variational Autoencoders

Adithya Vellal Eric Tay
Duke University
adithya.vellal@duke.edu, eric.tay@duke.edu

Abstract

The tasks of density estimation and data generation are becoming an increasingly important part of unsupervised machine learning as we seek to gain insights about the underlying structure of complex distributions using datasets that comprise a relatively small number of samples. In this paper, we employ the Variational Autoencoder (VAE), a deep learning latent variable model, to gain insights into the structure of the Fashion MNIST dataset. We explore the VAE's efficacy in the tasks of image reconstruction, latent space projection, synthetic data generation, and cross category interpolation.

1 Introduction

The MNIST dataset is a large database of handwritten digits that is commonly used for training and benchmarking various machine learning algorithms. However, as outlined by Xiao et al. [1], MNIST is so simple that it often fails to effectively represent model performance on most modern computer vision tasks. While most models achieve very high accuracy on MNIST, their performance does not generalize well to other datasets. Thus, in this project we use the Fashion-MNIST dataset, produced by e-commerce company Zalando in order to alleviate some of the common issues stemming from the traditional MNIST dataset. In order to augment this dataset's benefits and improve model robustness during the training stage, we wish to generate synthetic samples from the underlying distribution represented by the provided data [2]. We also seek to learn fruitful low-dimensional latent representations of this data (eg. presence of buttons, type of fabric, etc.) which will enable the generation of hybrid fashion products that give inspiration to fashion designers. We aim to accomplish these objectives through the use of a Variational Autoencoder (VAE), which employs a Bayesian approach to learn our latent variables a posteriori through a function parameterized by a neural network. One of the challenges we expect to encounter in pursuit of our secondary goal is that our learned latent space may not be continous enough to bridge the gap between extremely dissimilar categories such as sneakers and shirts. Meanwhile, other categories such as sneakers and sandals may be so similar already that there is very little room for novel product development.

2 Data

We use Zalando's Fashion-MNIST dataset, which consists of 70,000 28x28 grayscale images. 60,000 of these images belong to the training set, while the remaining 10,000 belong to the test set. There are 10 total labels, with integer labels of 0-9 as follows: {T-shirts/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot}. The data type is hence nominal discrete. This dataset is highly heterogeneous yet also extensive, making it a highly promising and intriguing choice for studying synthetic data generation with VAEs.

We use t-Distributed Stochastic Neighbor Embedding (t-SNE) [3] to project and visualize our raw data. This embedding method maximizes the similarity between our original and 2-dimensional

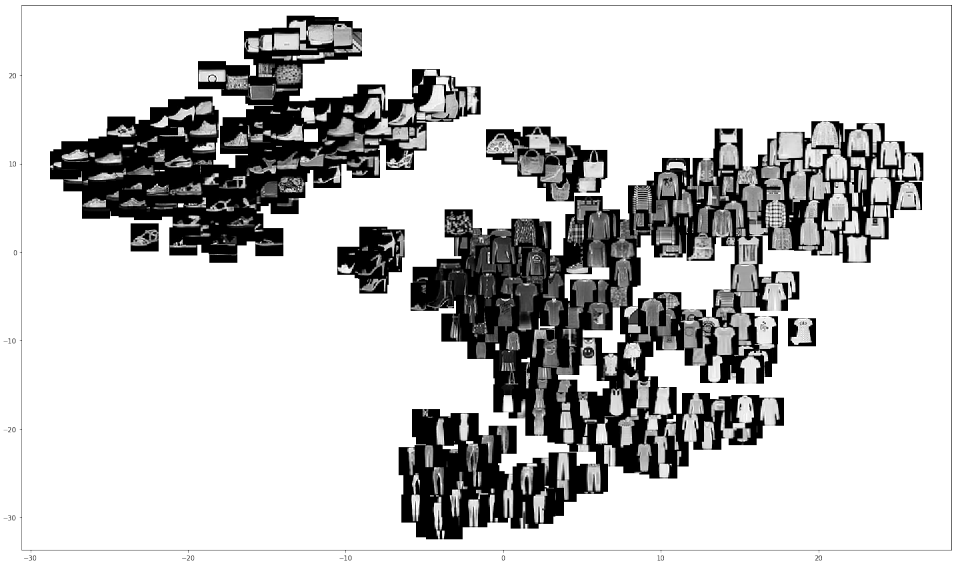

Figure 1: t-SNE Projection of Images Based on Raw Pixel Values

representation of points, thereby capturing the non-linear manifold structures present in our data. As expected, our projection primarily groups clothing items based on their color, as similarly colored items are close together in their pixel values. However, it cannot separate objects based on their latent structure, which could take the form of broad categories (eg. tops, bottoms, footwear, bags) or finer details (eg. pattern textures, brand logos, buttons, sleeves/trouser legs). It also becomes clear that simply interpolating pixel values will not produce fashionable hybrid products that are combinations of closely-related categories like Pullover, T-shirts, and Coat.

3 Model

3.1 Choice of Model

Given that we wish to learn the latent structure of image data, a natural model choice is a convolutional autoencoder (AE). An AE consists of two major components: an encoder and decoder. The encoder attempts to generate a low-dimensional representation of our input image x_i through a succession of convolutional and downsampling operations. Meanwhile, the decoder uses convolutional layers and subsequent upsampling operations to reconstruct $\tilde{x_i}$ from the latent representation. The network weights are trained using a pixelwise mean squared error (MSE) reconstruction loss, $||x_i - \tilde{x_i}||^2$, and we use a latent dimension size, k, of 128. If our trained AE can reconstruct x_i from our k-dimensional latent space with high fidelity, then we can conclude that our latent vector effectively encapsulates the information present in x_i. Thus, to generate new data points, we can hypothetically sample points from our latent space and decode them using our trained decoder.

However, in practice, solely optimizing our dimensionality reduction process for reconstruction loss leads to a highly irregular and discontinuous latent space. The high degree of freedom provided by the convolutional encoder and decoder networks enable the AE to overfit the latent space. Geometrically, each data point is encoded as a point in latent space, and points that are nearby in the latent space are not necessarily close together in the original data distribution. We address this problem by introducing the VAE framework, which regularizes training by forcing the learned latent representation to be a gaussian distribution that is similar to a k-dimensional unit gaussian. Encoding each data point as a continuous distribution ensures that nearby points in our initial distribution are also close together in the latent space. Additionally, this regularization forces our latent space to be more continuous, allowing us to generate hybrids as we explore the latent space.

3.2 Bayesian Derivation

The convolutional encoder, θ, and decoder, ϕ, parameterize the posterior latent distribution $q_\theta(z|x_i)$ and likelihood $p_\phi(x_i|z_i)$ respectively. In order to regularize the latent space as specified above, we place a prior $p(z) \sim N(\mathbf{0}, I_k)$ on our latent variables, allowing us to rewrite $p(z|x) = \frac{p(z)p(x|z)}{p(x)}$. The computation of the denominator $p(x)$ in this expression is intractable, so we utilize variational inference to approximate $p(z|x)$ with $q_\theta(z|x)$. The regularization term of our loss function is thus the 'dissimilarity' between $q_\theta(z|x)$ and the true posterior, $p(z|x)$, which we quantify using the KL-divergence: $\mathbb{KL}(q_\theta(z|x) \,||\, p(z|x))$.

Now we note the following (Appendix 6.1):

$$\begin{aligned}\arg\min_\theta \mathbb{KL}(q_\theta(z|x) \,||\, p(z|x)) &= \arg\min_\theta \Big[\mathbb{KL}(q_\theta(z|x) \,||\, p(z|x)) - \log p(x)\Big] \\ &= \arg\min_\theta \Big[-\mathbb{E}_{z\sim q_\theta(z|x)}[\log p(x|z)] + \mathbb{KL}(q_\theta(z|x) \,||\, p(z))\Big]\end{aligned}$$

Hence, given $p(x|z)$, we have an optimal encoder. However, since we must also approximate $p(x|z)$, we do so by noting that given an encoding, our decoder should minimize the reconstruction loss $\mathbb{E}_{z\sim q_\theta(z|x)}[\log p_\phi(x|z)]$. Putting the two terms together gives us the overall loss function $L(\theta, \phi_x)$:

$$-\mathbb{E}_{z\sim q_\theta(z|x)}[\log p_\phi(x|z)] + \mathbb{KL}(q_\theta(z|x) \,||\, p(z))$$

We set $q_\theta(z|x_i)$ and $p_\phi(x_i|z_i)$ to be multivariate gaussians with diagonal covariances for ease of implementation. This implies that the various latent dimensions are uncorrelated. While this is certainly not accurate, it is a trade-off that we make for the ease and efficiency of implementation. Additionally, the choice of $N(\mathbf{0}, I_k)$ as our latent space prior may appear to be an oversimplification. However, we note that any k-dimensional distribution can be generated by taking a set of k variables that are normally distributed and mapping them through a sufficiently complicated function p_ϕ[4].

3.3 Implementation Details

We refer to `vae-cnn.py` for the implementation of this model on our Fashion-MNIST dataset. We note the following details that are used in the code:

- Our latent posterior is parameterized by the learned parameters μ_x and Φ_x. We take the $\log$ of Φ_x for numerical stability and store all the diagonal entries in the k-dimensional vector Σ_x. Given that $p(z) \sim N(\mathbf{0}, I_k)$, we can show that $\mathbb{KL}(q_\theta(z|x) \,||\, p(z))$ takes the form $\frac{1}{2}\sum_{i=1}^{k}(e^{\Sigma_{x,i}} + \mu_{x,i}^2 - 1 - \Sigma_{x,i})$ (Appendix 6.2).
- For the reconstruction loss, we generally take the term out of the expectation due to computational efficiency. Repeating this computation over multiple epochs can be seen as taking a Monte Carlo estimate of the reconstruction.
- We justify our MSE reconstruction loss formulation by noting that the log-likelihood of a multivariate gaussian centered at the output values of our model is simply the pixelwise MSE. Adding in the KL-divergence term ensures regularity (Appendix 6.3).
- While we are able to take the gradient of the loss with respect to ϕ, we are unable to do so with respect to θ due to the expectation term being taken over a distribution of ϕ. To address this, we shall reparameterize z to take the form, $z_i = \mu_{x,i} + \sqrt{\Phi_{x,i}} * \epsilon_i$, $\epsilon_i \sim \mathcal{N}(0, 1)$ (Appendix 6.4). Hence, $z_i = \mu_{x,i} + e^{\frac{1}{2}\Sigma_{x,i}} * \epsilon_i$, enabling us to perform backpropagation. This is commonly referred to as the reparameterization trick [5].
- To better capture the spatial structure present in our dataset, we use convolutional layers in both our encoder and decoder. Our encoder consists of 4 convolutional layers with kernel sizes 2 and 3, and a stride of 2. The encoder ouputs a 1x1x512 volume which is then fed into a fully-connected layer to learn a 128-dimensional latent representation. Our decoder takes our latent representation z and uses one fully-connected layer to map it to a 512-dimensional vector. This is then "unflattened" into a 1x1x512 volume which is subsequently fed through 4 convolutional layers of kernel sizes 2-5, and strides of 2, to obtain our output image.

4 Results

4.1 Image Reconstruction

Our network converges after training for 50 epochs. To evaluate performance, we perform a visual comparison of the original images with our reconstructed images from our test set (Figure 2).

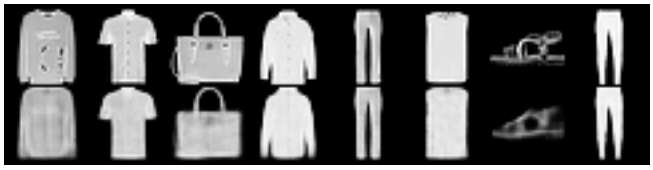

Figure 2: Original test images (top row) and their reconstruction (bottom row)

4.2 Latent Space Projection

We use t-SNE to visualize our datapoints in the latent space (Figure 3) and note that the model does a good job of grouping clothes based on category. It also appears to have learnt variables that allow it to discern fine details such as the difference between various types of footwear or the presence of sleeves. Unlike the initial t-SNE projection based on raw pixels, we see that this latent projection also captures the similarity between bags with and without handles. This plot indicates that our latent variables capture various defining features of fashion products.

One concern is that while some of the gaps between clusters have been bridged, the latent space is not completely continuous. In order to further investigate, we plot the values for the first two latent variables corresponding to images in category 0 (Figure 4). We notice that the values are still close to our prior of mean 0 and variance 1, meaning that our latent space is likely to be highly regular.

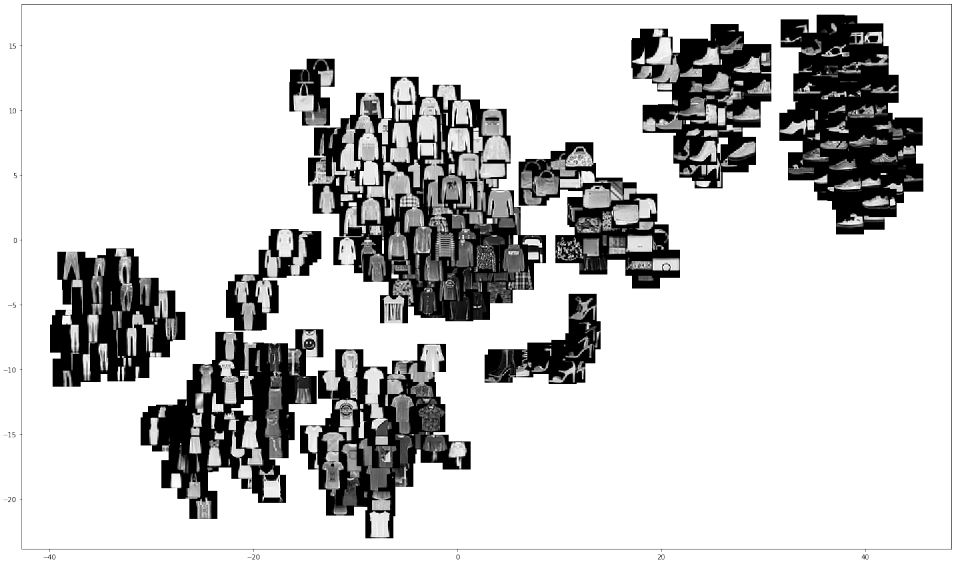

Figure 3: t-SNE Projection of Images Based on Learned Latent Representations

4.3 Data Generation Within Label Category

We take the mean latent representation across all training examples in a category to estimate the category's "center". Reconstructing this image gives us a sample of an "average" bag (Figure 5). We then sample with random noise around this mean to generate novel bags (Figure 6).

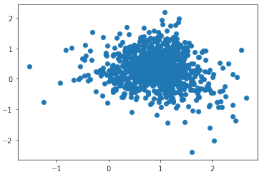

Figure 4: Plot of first two latent variables for category 0 images (T-shirts / tops)

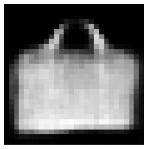

Figure 5: Image reconstructed from category mean

Figure 6: Generating new bag samples

4.4 Cross Category Interpolation

To generate hybrid images, we randomly pick two images and interpolate between them (Figure 7). We note that if we interpolate in the pixel space, we simply get weird overlapping fashion items. In contrast, if we interpolate in the latent space, we manage to create a hybrid object. In this case, we get a high heeled shoe with an interesting hole. We include other images we have generated to illustrate our ability to generate hybrid fashion products. This can be used by fashion designers who may want to look at the cross between closely-related products (Figure 8), or perhaps gain inspiration by crossing unexpected objects for interesting shapes/textures (Figure 9).

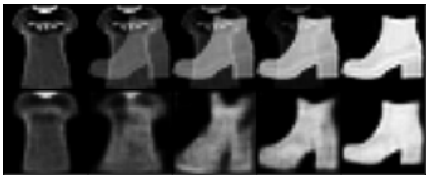

Figure 7: Pixel Interpolation (top) vs. Latent Interpolation (bottom) between a top and ankle boot

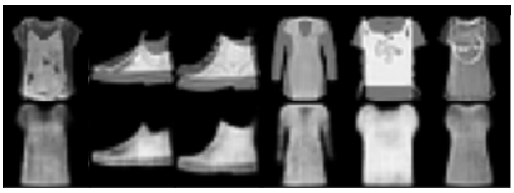

Figure 8: Crossing similar products (top) to obtain hybrids (bottom)

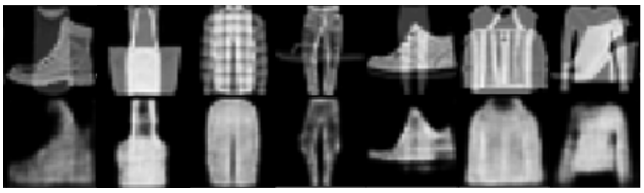

Figure 9: Crossing dissimilar products (top) to obtain hybrids (bottom). Rightmost picture depicts interesting texture.

5 Conclusion

We have successfully applied a probabilistic, bayesian deep learning model, the VAE, to our fashion MNIST data. Our model learned latent features of the products in a regular latent space, allowing us to reconstruct test images. Ultimately, we sucessfully generated synthetic data in order to make the dataset more robust and creating hybrid products for fashion designers. We expected to face more difficulty in crossing dissimilar fashion products, but these actually gave us interesting shapes and textures. Further extensions include analyzing the latent space we obtained from the data for possible insights or uses in downstream tasks like classification.

SUPPLEMENTAL MATERIALS

vae-cnn.ipynb: Main code that runs our algorithm for model training and visualization. Skip the cell which trains the model to use the pretrained model provided. (Jupyter Notebook)

vae.py: Code for initialization, training and testing the model. (Python Script)

visualization.py: Code for visualizing our data and results. (Python Script)

vae.torch: Trained Model (Torch file)

References

[1] H. Xiao, K. Rasul, R. Vollgraf (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747.*

[2] C. G. Turhan and H. S. Bilge (2018). Recent Trends in Deep Generative Models: a Review. *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, 574-579.

[3] L. van der Maaten and G. Hinton (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, Vol. 9, 2008, 2579- 2605.

[4] C. Doersch (2016). Tutorial on Variational Autoencoders. *arXiv preprint arXiv:1606.05908.*

[5] D. P. Kingma and M. Welling (2019). An Introduction to Variational Autoencoders. *Foundations and Trends in Machine Learning*.

6 Appendix

6.1 Proof of loss function

We wish to prove:

$$\arg\min_{\theta} \mathbb{KL}(q_\theta(z|x) \,||\, p(z|x)) = \arg\min_{\theta} \Big[-\mathbb{E}_{z\sim q_\theta(z|x)}[\log p(x|z)] + \mathbb{KL}(q_\theta(z|x) \,||\, p(z)) \Big]$$

Proof:

$$\begin{aligned}
\mathbb{KL}(q_\theta(z|x) \,||\, p(z|x)) &= -\mathbb{E}_{z\sim q_\theta(z|x)} \log \frac{p(z|x)}{q_\theta(z|x)} \\
&= -\mathbb{E}_{z\sim q_\theta(z|x)} \log \frac{p(x|z)}{p(x)} - \mathbb{E}_{z\sim q_\theta(z|x)} \log \frac{p(z)}{q_\theta(z|x)} \\
&= -\mathbb{E}_{z\sim q_\theta(z|x)}[\log p(x|z)] + \mathbb{KL}(q_\theta(z|x) \,||\, p(z)) + C,\ C \in \mathbb{R}
\end{aligned}$$

6.2 Proof of KL Divergence Term

Recall that our model learns a k-dimensonal mean vector μ_x and covariance matrix Φ_x. If Φ_x is diagonal, and $p(z) \sim N(\mathbf{0}, I_k)$, we will prove that

$$\mathbb{KL}(q_\theta(z|x) \,||\, p(z)) = \frac{1}{2} \sum_{i}^{k} (e^{\Sigma_{x,i}} + \mu_{x,i}^2 - 1 - \Sigma_{x,i}),$$

where Σ_x is a k-dimensional vector storing the log of diagonal entries of Φ_x.

Proof:

$$
\begin{aligned}
&\mathbb{KL}(q_\theta(z|x) \,||\, p(z)) \\
&= -\int q_\theta(z|x) \log \frac{p(z)}{q_\theta(z|x)} dz \\
&= -\mathbb{E}_{z\sim q_\theta(z|x)}\left(\log\Big(\frac{1}{\sqrt{(2\pi)^k}}\Big) - \frac{1}{2} z^T z - \log\Big(\frac{1}{\sqrt{(2\pi)^k |\Phi_x|}}\Big) + \frac{1}{2}(z-\mu_x)^T \Phi_x^{-1}(z-\mu_x) \right) \\
&= -\frac{1}{2}\log|\Phi_x| - \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big((z-\mu_x)^T \Phi_x^{-1}(z-\mu_x) - z^T z \Big) \\
&= -\frac{1}{2}\log|\Phi_x| + \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big((z-\mu_x)^T (z-\mu_x) + 2\mu_x^T z - \mu_x^T \mu_x - (z-\mu_x)^T \Phi_x^{-1}(z-\mu_x) \Big) \\
&= -\frac{1}{2}\log|\Phi_x| + \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big((z-\mu_x)^T (z-\mu_x) + \mu_x^T \mu_x - (z-\mu_x)^T \Phi_x^{-1}(z-\mu_x) \Big) \\
&= \frac{1}{2}(\mu_x^T \mu_x - \log|\Phi_x|) + \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big(tr\Big((z-\mu_x)^T (z-\mu_x) \Big) \Big) - \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big(tr\Big((z-\mu_x)^T \Phi_x^{-1}(z-\mu_x) \Big) \Big) \\
&= \frac{1}{2}(\mu_x^T \mu_x - \log|\Phi_x|) + \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big(tr\Big((z-\mu_x)(z-\mu_x)^T \Big) \Big) - \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big(tr\Big(\Phi_x^{-1}(z-\mu_x)(z-\mu_x)^T \Big) \Big) \\
&= \frac{1}{2}(\mu_x^T \mu_x - \log|\Phi_x|) + \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big(tr(\Phi_x) \Big) - \frac{1}{2}\mathbb{E}_{z\sim q_\theta(z|x)}\Big(tr(\Phi_x^{-1}\Phi_x) \Big) \\
&= \frac{1}{2}(\mu_x^T \mu_x - \log|\Phi_x| + tr(\Phi_x) - k) \\
&= \frac{1}{2}(tr(\Phi_x) + \mu_x^T \mu_x - k - \log|\Phi_x|) \\
&= \frac{1}{2}\sum_i^k (e^{\Sigma_{x,i}} + \mu_{x,i}^2 - 1 - \Sigma_{x,i})
\end{aligned}
$$

6.3 Proof of Reconstruction Loss

We shall prove that if we reconstruct each pixel using a Gaussian centered at the output values of our model, the log-likelihood of reconstruction can be represented by the MSE between our output values and input pixel values. Specifically, we wish to show that:

$$\log p_\phi(x_i|z_i) = C - D\sum_{j=1}^{784}(x'_{ij} - x_{ij})^2,\ C, D \in \mathbb{R},\ D > 0,$$

where z_i is the encoding of input x_i, and x'_i is the decoding of z_i, or the output of our model.

Proof:

$$
\begin{aligned}
\log p_\phi(x_i|z_i) &= \log p_\phi(x_i|x'_i) \\
&= \log \prod_{j=1}^{784} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_{ij}-x'_{ij})^2}{2\sigma^2}} \\
&= C - D\sum_{j=1}^{784}(x_{ij} - x'_{ij})^2,\ C, D \in \mathbb{R},\ D > 0
\end{aligned}
$$

6.4 Proof of Reparameterization

We know that if $\epsilon_i \sim N(0,1)$, $X = a + b\epsilon_i \sim \mathcal{N}(a, b^2)$.

We can hence write $z_i \sim \mathcal{N}(\mu_{x,i}, \Phi_{x,i})$ as $z_i = \mu_{x,i} + \sqrt{\Phi_{x,i}} * \epsilon_i$.