# Seminar on Scientific Machine Learning: Likelihood-based Generative models: VAEs and NFs

Elias Huber, Sam Rouppe Van der Voort

August 24, 2025

## Abstract

This Document summarizes the Seminar talk on "Likelihood-based Generative Models: VAEs and NFs" of the Scientific Machine Learning Seminar. Here, we first introduce Likelihood-based models in a general scope. We then continue with deriving the theory behind Variational Autoencoders and Normalizing Flows. We also give small example implementations for both to give a proof of principle. We then proceed with two experimental tasks, where we use the presented methods to train models for learning the distribution generated by a QR-code and an image generation model that uses Normalizing Flows and Variational Autoencoders together to generate and classify images of fruit. Finally, we conclude our presentation by discussing the possible applications in physics. All code examples can be found on our GitHub repository [SRvdV25], and this report can be found at [HRVdV25] with animated GIFs for better illustration.

## 1 Introduction

In recent years, Likelihood-based generative models have had a significant impact in fields like computer vision [BK21] and natural language processing [Zha25]. Two foundational methods of this field are Variational Autoencoders (VAEs) [KW13, KW19] and Normalizing Flows (NFs) [RM15, PNJR+21].
In this article, we will present and discuss both of these methods, provide implementation examples, and discuss their applications in physics.

## 2 Likelihood-based Generative Modeling

Likelihood-based generative modeling deals with the issue of generating samples that match the characteristics of a given dataset. Mathematically, we have given a dataset $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ that follows an underlying data distribution $p(\mathbf{x})$. To sample and evaluate new data, we want to find a distribution $p_\theta(\mathbf{x})$ that approximates the underlying true distribution $p_($**x**$)$ that the data follows. For this, we assume some approximation function $p_\theta(\mathbf{x})$ and then optimize the selection of the parameter $\theta$ such that it matches the original distribution the best. This is done by maximizing the likelihood

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i)$$

that we observe our data, given our approximation of the probability distribution $p_\theta(\mathbf{x})$ with parameters $\theta$. In practice, we model our approximation distribution $p_\theta(\mathbf{x})$ with neural networks and minimize the negative log-likelihood

$$\begin{aligned}\hat{\theta}_{MLE} &= \arg\min_\theta J(\theta, \mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(n)}) \\ &= -\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)}),\end{aligned}$$

to find the optimal weights of the neural network. We minimize the log-likelihood due to numerical stability. The parameter optimization process is then done using gradient descent-based algorithms.

## 3 Variational Autoencoders

### 3.1 Theory

Variational Autoencoders (abbreviated VAEs) introduced by Kingma et al in [KW13], are conceptually similar to Autoencoders (abbreviated AE). The main idea is to encode data in a lower-dimensional latent space and then to recreate or create new data from the latent space. Practically, both architectures achieve this by consisting of an encoder and a decoder. Unlike AEs, which encode each data point to a point in the latent space, VAEs encode each data point to a distribution in the latent space. This is achieved

by the encoder and decoder approximating conditional probability distributions. In particular the probability of a latent state given a data point $p(z|x)$, and conversely the probability of a data point given a latent state $p(x|z)$. For VAEs both distributions are approximated with neural networks $p(z|x) \approx q_\phi(z|x)$ for the encoder and $p(x|z)$ for the decoder we approximate $p(x|z) \approx p_\theta(x|z)$ with parameters $\phi$ and $\theta$ respectively. For the latent space to be a distribution $q_\phi(z|x)$ outputs the parameters of the distribution of the latent space. Commonly gaussian with a mean vector $\mu_\phi$ and a standard deviation vector $\sigma_\phi$.

#### 3.1.1 ELBO

As discussed before VAEs are a subclass of likelihood based modeling. So we want to maximize the log-likelihood to obtain the loss function for the VAEs. An exact solution is not tractable but we can obtain a lower bound. This results in the ELBO, the loss function to achieve the approximations of the conditional probability distributions. The ELBO is the expectation value, wrt. samples from the encoder probability distribution $q_\phi$ expressed as follows:

$$\begin{aligned}\text{ELBO}(x) &= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z|x)}\right] \\ &= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]}_{\text{Reconstruction}} - \underbrace{D_{\text{KL}}(q_\phi(z|x)\,\|\,p(z))}_{\text{Regularizing}}.\end{aligned}$$

We see that the ELBO can be split up into a reconstruction term and a regularizing term. The reconstruction term should ensure a good reconstruction and depends on the assumed distribution of the target. For example, the reconstruction term becomes the MSE for gaussian assumption or BCE loss for Bernoulli assumption. While the regularizing term enforces the encoder distribution to resemble the assumed prior in latent space, which for the vanilla VAE case is standard gaussian.

#### 3.1.2 Reparameterization trick

For calculating the ELBOs reconstruction term in practice $\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$, we rely on samples $z$ from the learned distribution $q_\phi(z|x)$. The sampling however is not differentiable wrt. $\phi$. To circumvent this problem, Kingma et al. [KW13]) proposed a so called reparameterization trick. The transformation $z = \mu_\phi + \sigma_\phi \odot \epsilon$ in terms of the distribution parameters $\mu_\phi$ and $\sigma_\phi$ from $q_\phi(z|x)$ makes the sampling independent of $\phi$. The stochastic variable $\epsilon$ is sampled form a standard normal distribution $\mathcal{N}(0, \mathrm{I})$ and due to the properties of Gaussians, the distribution of $z(\mu_\phi, \sigma_\phi, \epsilon)$ will still be normal $\mathcal{N}(\mu_\phi, \sigma_\phi)$. So the distribution of $z$ remains unchanged and the transformation allows samples of $z$ to be differentiated wrt. $\phi$ since the stochasticity now is in $\epsilon$.


Figure 1: Latent distribution of encoded images, colored by label.

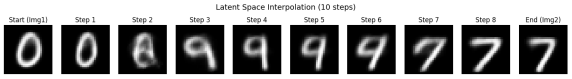


Figure 2: Linear interpolation between MNIST images.

### 3.2 Implementation: MNIST

For a showcase of VAEs we implemented a VAE for generating new MNIST-like images. We did this with torch by using feedforward neural networks for the encoder and decoder networks, approximating $q_\phi(z|x)$ and $p_\theta(x|z)$. We choose a latent dimension of 2 for visualization shown in Figure 1. Further more, we explored the continuity of the latent space by interpolating between two encoded images, decoding along the way. The result is shown in Figure 2 or as Gif on our GitHub repository [SRvdV25]. Finally, we also generated new MNIST-like images, see Figure 3.

## 4 Normalizing Flows

### 4.1 Theory

Normalizing flows have a different approach than VAEs presented before. Instead of learning our target distribution directly, we learn a transformation $f$ from a base distribution $p_Z(\mathbf{z})$ to our target

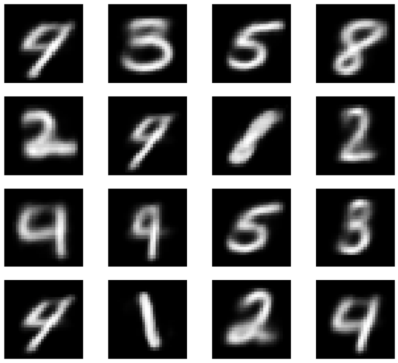

Figure 3: Novel MNIST images generated with trained VAE.

distribution $p_X(\mathbf{x})$. This enables us to efficiently sample from our target distribution, by sampling from the base distribution and transforming the sample to the target space. Mathematically, we utilize the change of variables formula

$$p_X(\mathbf{x}) = p_Z(f(\mathbf{x}))\,|\det J_f(\mathbf{x})|^{-1}$$

to find our transformation. Here, $f(\mathbf{x})$ is a bijective function $f : \mathbb{R}^d \to \mathbb{R}^d$ that transforms both equally dimensional probability spaces into each other. The significant part of this equation is the calculation of the normalization constant, given by the determinant of the Jacobian matrix of the transform $|\det J_f(\mathbf{x})|^{-1}$. This restrains our implementation of the transformation function $f(x)$ to be invertible and differentiable. We thus need to use invertible and fully differentiable neural networks to implement the transformation.

#### 4.1.1 Coupling Flows

One prominent example of a normalizing flow implementation is the so-called coupling flow. Here, we split our input data in two parts. While we leave the first part of the input dimensions unchanged, we use them as input to a neural network that outputs a transform that acts on the second part. This neural network is also called a coupling network. The schematic structure of such a coupling network can be seen in Figure 4. This kind of flow is useful since the Jacobian of the transformation takes the form of a lower triangular matrix, which reduces the complexity of the determinant calculation from $\mathcal{O}(d^3)$ to $\mathcal{O}(d^2)$, where $d$ is the dimensionality of the distributions.

#### 4.1.2 Composition of Flows

To enhance the expressiveness of normalizing flows, it is possible to concatenate multiple normalizing flow transforms, to divide the task of finding the proper transform into multiple subtasks

$$f = f_K \circ f_{K-1} \circ \cdots \circ f_1.$$

The determinant, necessary for normalization then can be calculated with the relation

$$\det J_f = \det J_{f_K} \cdot \det J_{f_{K-1}} \cdots \det J_{f_1} = \prod_{k=1}^{K} \det J_{f_k}.$$

This trick greatly increases the expressive power of normalizing flows while keeping computational cost low, by implementing simple transforms like the previously shown coupling flows and concatenating them to achieve higher expressiveness.

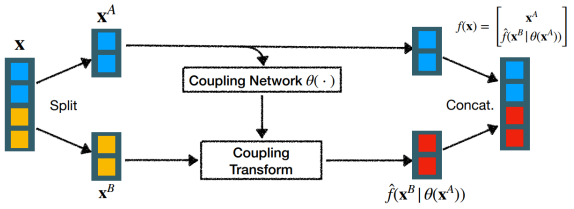


Figure 4: Schematic of a coupling flow [BK21].

### 4.2 Implementation

For a proof of concept, We used normalizing flows to learn a double-peaked gaussian in two dimensions. For that, we defined the standard 2D gaussian as the base distribution. For the normalizing flow, we concatenated eight coupling transforms that use affine transformations as coupling networks. The implementation was done in python, using the nflows library with torch. The base, target and generated distribtions are shown in Figures 5. The code can be found in the notebook NFexample.

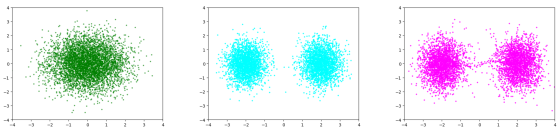

Figure 5: From left to right: Samples from base, target and generated distributions.

## 5 Experiments

To test both of these methods, we implemented two case studies where we tried to bring both methods to their limits.

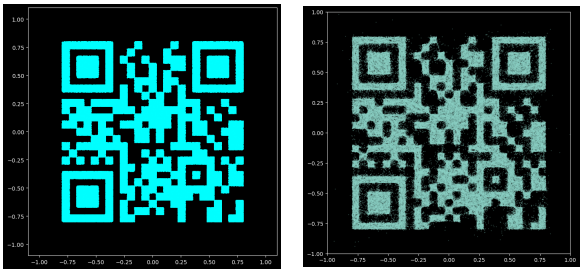

Figure 6: Left: Target QR code. Right: Generated samples using flow matching.

## 5.1 Learning the distribution of a QR code

In an effort to probe the expressiveness of normalizing flows, we tried to learn the distribution that is obtained by transforming the QR-Code of our Github repository into an uniform distribution (see Figure 6). We first tried to use different types of normalizing flows to learn the transformation from a standard 2D Gaussian to the underlying distribution of the QR-Code. We observed that the method failed to capture the fine structure of the QR code pixels. We thus modified our approach to use a more advanced and computationally more efficient approach to the problem: Flow matching. Here, instead of finding discrete transforms, we try to find a continuous velocity field that changes between the times 0 and 1, to push our samples forward in time using an ODE integrator. Note that this method has close ties with the previous seminar on neural ODEs. The resulting generated distribution looked more like a QR code but was still not scannable. The sharp edges of the pixels were limiting the performance. We thus extended our transformation to also learn the Fourier features of the target distribution, to increase the precision for high frequency features like the sharp edges of the pixels. This led to the ability to recreate the QR code distribution with high accuracy. The result is shown in Figure 6. An animation of the Flow matching process can be found in our GitHub repository [SRvdV25].

## 5.2 Image generation of Fruits

For further showcase of VAEs we implemented a VAE for generation of fruit images. We used a CNN-based architecture for the encoder and decoder. The data used was around 1400 images from Kaggle [Zha20]. The three classes of apples, kiwis and bananas had approximately equal share. The generation of new samples was not sastisfying at first, so we tried to improve our model by adding normalizing flows to the latent distribution to improve the flexibility of the encoder $q_\phi$.

### 5.2.1 Adding NFs for a more expressive latent space

The idea behind adding NFs, is that this approach lifts the limitation of the latent space being gaussian, which thus increases expressivity. Specifically we implemented inverse autoregressive flows (IAFs), which were suggested for VAEs by Kingma in [KSJ+16]. The IAF flow expression is $z^{(t)} = \mu^{(t)}(z^{(t-1)}) + \sigma^{(t)}(z^{(t-1)}) \odot z^{(t-1)}$. Notice that the form of the transformation is similar to the reparameterization trick, here however we perform flows up to T times, since we can compose multiple flows. This flow formulation has desirable properties for a normalizing flow as it is invertible and yields a triangular Jacobian. Visually, we did not see an improvement, and propose that we need more data or a pretrained visual model for the VAE to properly learn how to generate fruit images.

### 5.2.2 Latent space exploration

Similar to the example of MNIST, we explored the latent space of our models. We did this for both the pure VAE with CNN architecture and the VAE with additional normalizing flows in the latent space. To explore the latent sapce, we linearly interpolated between encoded images. Here the latent dimension however is much greater than for the MNIST example, where we have set it to 128 and 64 respectively. An example of such a latent space exploration can be found in Figure 7. Observing the latent sapce, we see that the VAE

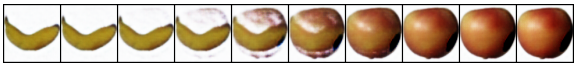

Figure 7: Linear interpolation between Banana and Apple.

struggles with recreating high quality, and coherent images. It seems however to still be able to separate between fruit classes because when trying to sample from the neighborhood of an encoded training data image, we yield blurrier versions of the encoded image, not different fruits.

# 6 Applications in Physics

Likelihood-based generative models are used predominantly in two different modes. We either want to generate new samples that weren't in the training set, or we want to evaluate the probability of new samples we observe.

### 6.1 Generating new samples

VAEs and NFs are both used for generative models, even though they are not state-of-the-art in some cases. Application fields include chemical design [GBWD+18] and generation of rare galaxy images for telescope calibration [RLM+16].

### 6.2 Evaluating samples

Both VAEs and NFs are used to evaluate sample probabilities. Since VAEs only evaluate the ELBO, which gives an lower bound, they are mainly used for anomaly detection [PBC+20]. NFs can also be used to efficiently evaluate more exact probabilities, as they are necessary for Markov Chain Monte Carlo methods [RHC21].

## 7 Summary

We gave an introduction to likelihood based generative modeling, discussing the mathematical framework as well as the concrete application form of the problem. We then introduced VAEs and Normalizing Flows, as prominent methods of likelihood based generative modeling. Defining features of the methods being that VAEs approximate the target distribution through an encoder decoder structure and Normalizing flows learn a transformation from a simple distribution to the target distribution. Our examples include learning the MNIST dataset and a 2D two-peaked Gaussian. In an effort to test the limits of these approaches we furthermore studied more challenging applications that include learning the distribution of a QR code and learning the distribution behind images of fruit. To solve these cases, we tired different architectures and approches, combining knowledge from previous seminar talks. Finally, we gave examples on where these methods are used in scientific research in physics.

## References


[BK21] Marcus A. Brubaker and Ullrich Köthe. Normalizing flows and invertible neural networks in computer vision: Cvpr 2021 tutorial. https://mbrubake.github.io, 2021. CVPR 2021 Tutorial.

[GBWD+18] Rafael Gomez-Bombarelli, Jennifer N. Wei, David Du, Miquel Gimeno, Jorge Zepeda-Cervantes, Md Ahmed, Benjamin Sanchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

[HRVdV25] Elias Huber and Sam Rouppe Van der Voort. Seminar on scientific machine learning: Likelihood-based generative models: Vaes and nfs. https://physics-master-computational-physics.github.io/, 2025. Accessed: 2025-08-24.

[KSJ+16] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934*, 2016.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[KW19] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[PBC+20] Adrian Alan Pol, Victor Berger, Gianluca Cerminara, Cécile Germain, and Maurizio Pierini. Anomaly detection with conditional variational autoencoders. *arXiv preprint arXiv:2010.05531*, 2020.

[PNJR+21] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

[RHC21] Alessandro Roggero, Christian Hackstein, and Joseph Carlson. Normalizing flows for microscopic many-body calculations: An application to the nuclear equation of state. *Physical Review Letters*, 127(6):062502, 2021.

[RLM+16] Siamak Ravanbakhsh, Francois Lanusse, Rachel Mandelbaum,

Jeff Schneider, and Barnabas Poczos. Enabling dark energy science with deep generative models of galaxy images. *arXiv preprint arXiv:1609.05796*, 2016.

[RM15] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[SRvdV25] Elias Huber Sam Rouppe van der Voort. Scientific_machine_learning_for_physics_seminar. https://github.com/Physics-Master-Computational-Physics/Scientific_Machine_Learning_for_Physics_Seminar, 2025. Code and Jupyter notebooks for the "Scientific Machine Learning for Physics" seminar.

[Zha20] Edward Zhang. Fruit recognition dataset. Kaggle Dataset, 2020. Accessed: 20.06.2025.

[Zha25] Wayne Xin Zhao. A survey of large language models, 2025.