A reliable implementation of the Variational Autoencoding Neural Operator (VANO)

Matthew Lowery, Ramansh Sharma

1 Introduction

For this final project, we chose to implement VANO, the Variational Neural Operator (VANO; [**?**]). We considered it an interesting endeavor in so much as our research deals with operator learning and working with functional data, versus most ML methods that are limited to finite dimensional vectors. Additionally, VANO is a probabilistic model that performs variational inference which means it aligns itself well within the subjects covered in the Probabilistic Machine Learning class.

VANO is first a variational autoencoder (VAE), but adapted for functional datasets, which are a new paradigm for the field of machine learning (ML). VANO learns a smooth distribution over functions that can be sampled from to generate new functions, in a *generative* capacity. In addition, the latent distribution's dimensionality is reduced significantly as compared to the input, which makes a useful, compact, and accurate *summary* of the functional data that is cheaper to store and use. Among many applications, one can use the latent distribution to perform anomaly detection.

While classical methods such as k-means and PCA can encode any data, which may include function evaluations, they are not necessarily able to capture complex, physical, and non-linear relationship between the input data points. Moreover, they are far inferior to the highly-nonlinear approximation that deep learning methods is capable of. Moreover, classical methods cannot learn a probabilistic distribution which can be sampled from to generate new examples.

We initially thought VANO was tailored to learn operators; essentially, it encodes a given input function, samples from the latent distribution and uses the decoded representation as a proxy for the output function of the operator. While VANO potentially has this capability, **it currently focuses on learning the identity operator of the functions just as standard VAEs do with feature vectors, images, and other datasets**. We utilize the model in our work in this way, taking the input functions from some of the canonical operator learning datasets and learning their underlying distribution.

2 Motivation

The operator learning problem involves learning an approximation map between a given set of input and output functions [**?**, **?**] which can then be evaluated on previously unseen examples of functions. While machine learning techniques are readily useful with a large variety of architectures for this problem, more traditional scientific computing techniques are not. Traditional techniques such as principal component analysis (PCA) and proper orthogonal decomposition (POD) are by themselves not expressive enough for large scale datasets or difficult operator learning problems (ones that

involve mix of global/local information in the PDE, oscillatory problems, etc.). However, these techniques are useful in unsupervised models such as variational autoencoding neural operators (VANO) [**?**].

Without loss of generality, we describe the operator learning problem here. Let $U = \{u_i\}_{i=0}^{N_u}$ and $V = \{v_j\}_{j=0}^{N_v}$ be two sets of functions. The operator $\tilde{G} : \mathcal{U} \to \mathcal{V}$ is the continuous operator where $u \sim \mathcal{U}$ and $v \sim \mathcal{V}$. Discretely, let $u : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ be an input function and $x \in \mathbb{R}^{d_1}$ and $y \in \mathbb{R}^q$ be points where $d_1, d_2, q \in \mathbb{N}$. While it is not necessarily true, the output function v is usually also a map as such $v : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$. Using a machine learning method, one then approximates the operator $G : U \to V$ by optimizing parameters with a loss function.

3 VANO

VANO uses a traditional VAE to learn a reconstruction representation of *functional data*. It is an unsupervised machine learning model to generate high quality functional data, something that is limited in other models right now. It does so by first passing the functional data through an encoder network. This network is agnostic of any machine learning based architecture; popular choices include multilayer perceptrons (MLPs) or convolutional neural networks (CNN). Let the encoder map by denoted as $\mathcal{E} : \mathcal{U} \to \mathbf{z}$, where $\mathcal{X}$ is the space of the input function and $\mathbf{z}$ is the *latent representation*. In practice, $\mathbf{z}$ is a finite dimensional representation; $\mathbf{z} \in \mathcal{R}^n$. We experiment over the same range of n as [**?**]. The input function $u \sim \mathcal{X}$ is used at evaluated locations; $\mathbf{u} \sim \mathcal{U} \in \mathcal{R}^m$. To make the autoencoder variational, VAEs and VANO use the reparameterization trick to ensure that the mean and variance w.r.t. which the latent distribution are sampled are able to be backpropagated through by automatic differentiation. This is a common trick and is done as follows; given a mean μ and variance σ (that are being learned by the model), the latent distribution is sampled from these parameters as

$$\mathbf{z} = \mu + \sigma \, \epsilon, \tag{1}$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ is the standard normal distribution. In order to make sure the network does not learn the trivial variance of 0, in addition to the standard mean squared loss for reconstruction, the Kullback-Leibler (KL) divergence term is used to regularize the latent space to make it as close to a standard normal distribution as possible. This ensures stability in the model's performance. The KL loss is added as evidence lower bound (ELBO) in the model's loss function. The sample of the latent representation $\mathbf{z}$ (using the reparameterization trick) is then passed to the decoder network of the architecture. This architecture is similarly unconstrained for the architecture choice as the encoder. An appropriate architecture is decided based on the sampling grid of the input and output functions. If the functions are evaluated on a regular cartesian grid, a CNN is an appropriate choice. Otherwise, a graph neural network or a standard MLP is suitable. We denote the decoder map as $\mathcal{D} : \mathbf{z} \to \mathcal{V}$, where $\mathcal{V}$ is the space of the output functions (in the case of VANO, it is the same space as that of the input function). Once trained, the decoder component of the network can generate new functions in the latent coordinate space which can be evaluated anywhere in their domain. We reuse Figure **??**, borrowed from [**?**], which shows the encoder and decoder transformations on a function. The variational family of the latent space is chosen to be multivariate Gaussians with diagonal covariance.

Next, we describe the implementation of VANO in this project.

3.1 Implementation

In this project, we use the Jax machine learning framework [**?**]. While other machine learning frameworks such as PyTorch and Tensorflow exist, we pick Jax because of its superior modularity, composable functions, and functional perspective. In our `jax_networks.py` module, we provide implementations for common machine learning modules such as linear layers, MLPs, CNNs, as well as models relevant to our work such as DeepONets. Our main implementation of the VANO architecture is in this same module. In the VANO module, we split the output of the encoder MLP into the trainable mean and log-variance components. These two are then used to sample the latent distribution with the reparameterization trick. Subsequently, the decoder is composed of two MLPs; one to encode the spatial information where the ouptut function is evaluated, and the other to encode the sampled latent distribution. The two MLP outputs are concatenated and then passed through a final decoder MLP. The loss function is implemented in the individual files for the three problems we test on; `main_burgers.py`, `main_darcy_tri.py`, and `main_advection.py`.

3.2 GitHub

Our code base can be found at the following github link: `https://github.com/mwl10/generative_nos`.

4 Results

4.1 Dataset

In this project, we work with the following three datasets taken from [**?**]:

1. Burgers:

   The one dimensional Burgers' equation with periodic boundary conditions where $\nu = 0.1$ is the viscosity. The operator learning problem for VANO in this case is recovering the input function at the initial condition; $u_0(x)$.

   $$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, x \in (0,1), t \in (0,1]. \tag{2}$$

   The initial conditions for the training set of functions are generated according to a random distribution $\mu = \mathcal{N}(0, 625(-\Delta + 25\mathbf{I})^{-2})$. The spatial resolution is done with 128 grids.

2. Advection:

   The advection equation with periodic boundary conditions. The initial condition is chosen as a square wave. The mesh is chosen to be $40x40$ in space and time. VANO learns the initial condition $u_0(x)$.

   $$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0, x \in [0,1], t \in [0,1]. \tag{3}$$

3. Darcy flow:

   Darcy flow (or variable coefficient Poisson's equation) is our final dataset. A Gaussian process is used to generate the boundary conditions for each boundary on the triangular domain.

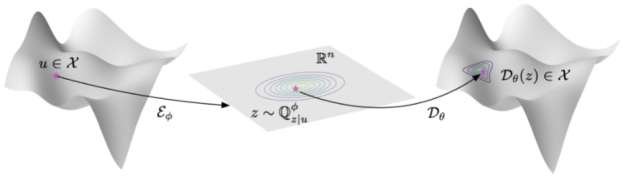

Figure 1: Figure 1 from VANO.

$$-\nabla \cdot (K(x,y)\nabla h(x,y)) = f,\ (x,y) \in \Omega, \tag{4}$$

where K is the permeability field, h is the pressure, and f is a source term which can either be a constant or a space-dependent function. **Note**: Darcy flow has 4 variations, (1) rectangular domain with continuous permeability field, (2) rectangular domain with piecewise constant permeability field, (3) triangular domain, and (4) triangular domain with a notch. We use the variation with the triangular domain with no notch for our study. VANO learns the input function of the permeability field.

5 Results

All our MLP networks are three layers wide, with 64 nodes each, using the leaky-ReLU activation function. We use the Adam optimizer to train the network with an inverse time decay learning rate schedule. The peak learning rate is chosen to be $1e-3$ with a decay rate of 0.5. We train over 50000 epochs. We go over the latent dimensions $n = [8, 16, 32]$ for our experiments.

1. Burgers' equation

   In Figures **??** and **??**, we present relative L_2 and L_∞ results over the training and test sets for the Burgers equation.We observe that VANO achieves a low enough relative error recovering the input function, on the order of magnitude of $1e-2$. The model is also agnostic in this problem to the latent dimension used n. This is probably due to the problem being one-dimensional. In higher dimension, we expect a higher n to approximate the function better with a richer latent distribution.

2. Advection equation

   Figures **??** and **??** show the relative errors on the Advection equation. We see similar VANO perform similarly in this problem as well. Since the problem is one-dimensional, the errors remain constant w.r.t. increasing n. We also observe a generally better order of error in this problem, $1e-3$. Further experiments are required to push the VANO error down further on space-time problems, potentially with a different encoding for the temporal component.

3. Darcy flow

   Lastly, Figures **??** and **??** show the relative errors on the Darcy flow problem on a triangular domain. While the order of error in this problem also goes down sufficiently low, $1e-3$, we observe that increasing n does not decrease the errors on either the training or test sets. Therefore, either a richer latent representation of the input functions is unnecessary in the given datasets, or a different VANO architecture is required to capture more spatial information.

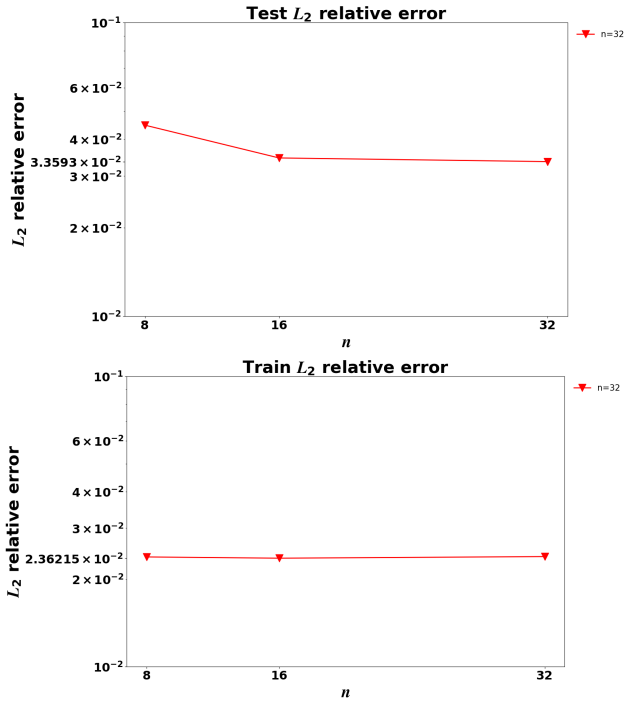

Figure 2: Relative L_2 errors on the training and test sets for the Burgers' equation.

One way to improve the encoder architecture in this regard is to use other standard networks (CNNs, graph neural networks) to encode spatial information better. We leave this for future experiments on these datasets.

6 Conclusion and Future Work

In the future, the important next step for VANOs is to apply the model to learn PDE operators and functional mappings other than the identity operator. Neither this work nor the original authors have done this as far as we know. We emphasize the ability of this model to learn the identity operator on functional data should not be understated because it implies a very useful capability. This is because functions are agnostic of their discretization in general, and accordingly, VANO can be used on datasets of any resolution (by using a set of evaluation points in the decoder MLP different from the points used in the input function evaluation). In particular, images can be thought of as functions and thus VANO provides an ability to train generative models that aren't particular to image resolutions. We imagine many different generative models will be adapted to work on functional data in the future with this methodology such as diffusion models and normalizing flows. In fact, recent papers on these subjects are tangentially approaching this idea [**?**].

References

[Kingma and Welling2022] Kingma, D. P. and Welling, M. (2022). Auto-encoding variational bayes.

[Li et al.2021] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2021). Fourier neural operator for parametric partial differential equations.

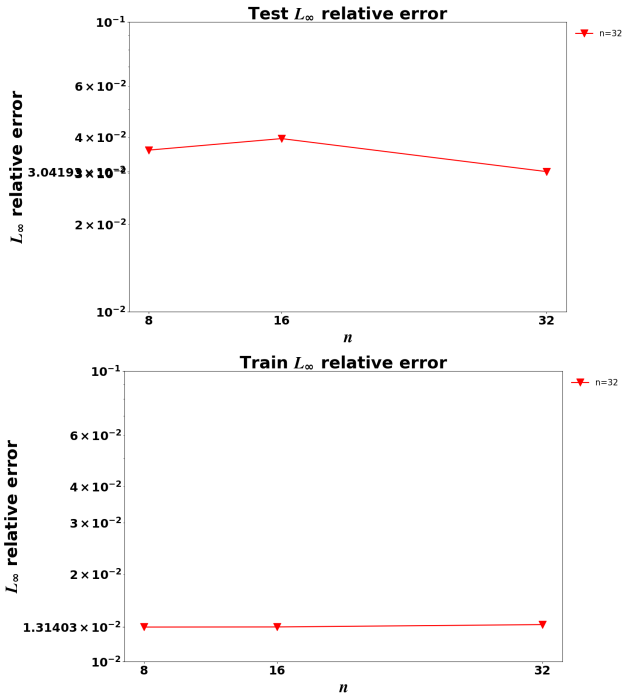

Figure 3: Relative L_∞ errors on the training and test sets for the Burgers' equation.

[Lu et al.2021] Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. (2021). Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229.

[Lu et al.2022] Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. (2022). A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778.

[Seidman et al.2023] Seidman, J. H., Kissas, G., Pappas, G. J., and Perdikaris, P. (2023). Variational autoencoding neural operators.

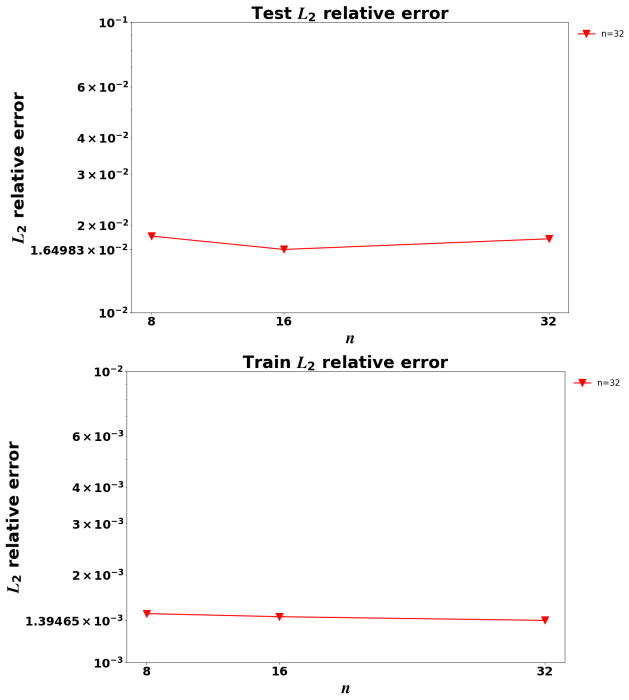

Figure 4: Relative L_2 errors on the training and test sets for the Advection equation.

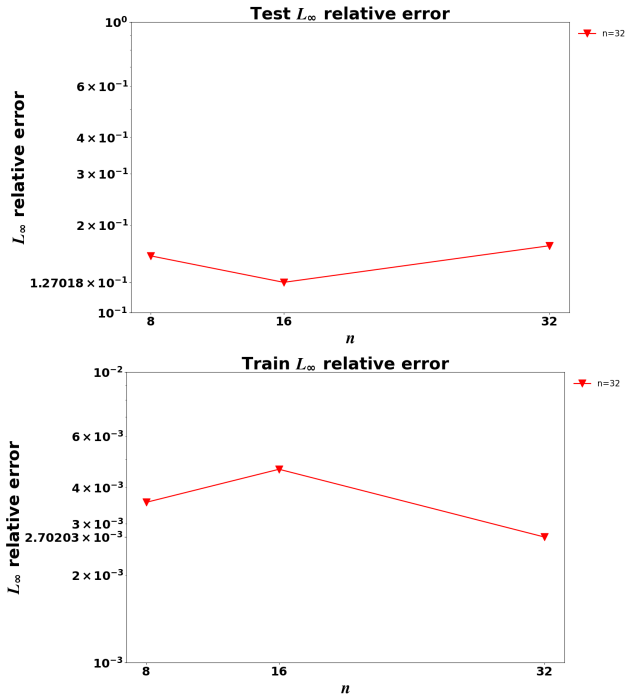

Figure 5: Relative L_∞ errors on the training and test sets for the Advection equation.

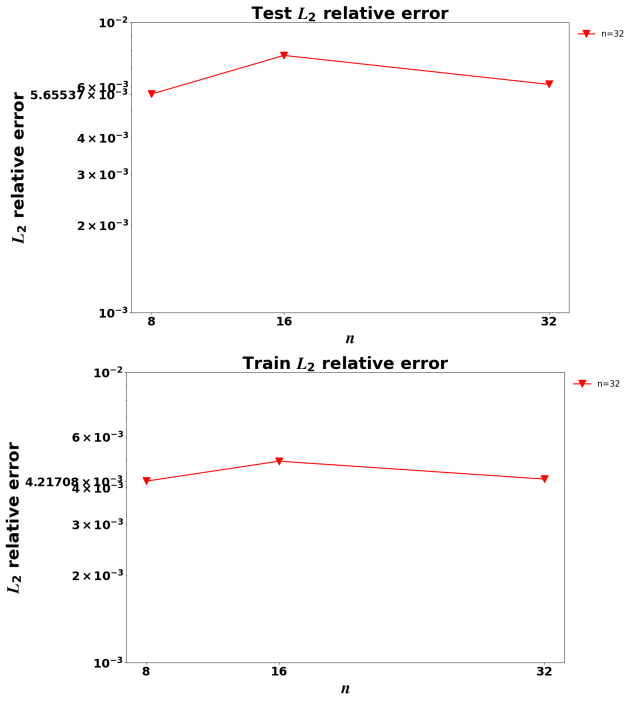

Figure 6: Relative L_2 errors on the training and test sets for the Darcy flow equation on the triangular domain.

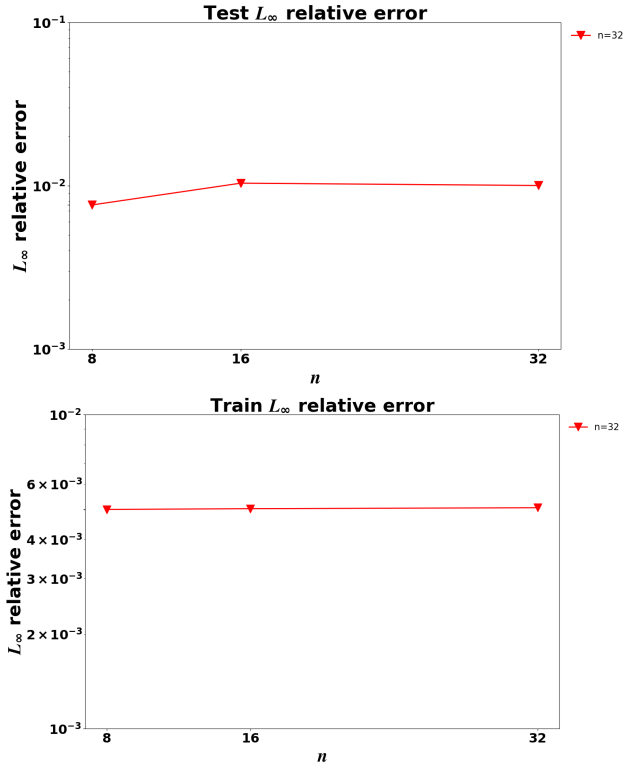

Figure 7: Relative L_∞ errors on the training and test sets for the Darcy flow equation on the triangular domain.