KAN-VAE: Combining Kolmogorov-Arnold Networks with Variational Auto-Encoders

Alireza Nik
SimulaMet & Oslo Metropolitan University
Oslo, Norway
alirezah@oslomet.no

***Abstract*—In this paper we introduce a new architecture in generative modeling. We integrate Kolmogorov Arnold Networks (KANs), a recently introduced alternative to Multi-Layer Perceptrons (MLPs), with Variational Autoencoders (VAEs). We replace MLPs in both encoder and decoder of VAEs with KAN layers. A comprehensive comparison of the performance of our proposed model, KAN-VAE, against a VAE baseline is conducted on the MNIST dataset. We hypothesize that this combination will improve the generative capabilities and parameter efficiency of VAEs. This work aims to bridge the gap between traditional VAEs and the new neural network architectures, potentially introducing promissing avenues for more efficient generative models.**

Keywords: Kolmogorov-Arnold Networks, Variational Autoencoders, Machine Learning, Nonlinear Representation Learning, Multi-layer Perceptron

The codes for this experiment are publicly available on https://github.com/ds-anik/KAN-VAE.

I. Introduction

Generative modeling has made significant progress in the past few years. Especially in domains such as computer vision and natural language processing, we have witnessed jaw-dropping models recently due to their impressive performance. Variational Auto-encoders (VAEs) [1] is among these models because of their ability to learn sophisticated patterns and relationships within the input data and generate realistic samples. Since their introduction, VAEs have shown promising performance in various domain specifically in feature extraction and representation learning [2].

Multi-Layer Perceptrons (MLPs) have been widely used within the VAEs' encoder and decoder components. This is due to their comparative performance in encoding the input data to the latent space and reconstructing it back, no matter how complex the relationships between the data are. However, despite this amazing performance, MLPs often require large numbers of parameters and deep network architectures in VAEs. From the training and inference efficiency point of view, this is not a positive point [3].

Recently, a new novel architecture has been introduced in the AI community that its authors claimed is a potential alternative to MLPs. Kolmogorov-Arnold Networks (KANs) [3]. The experiments in the original paper show significant improvements in interpretability and parameter efficiency compared to MLPs. This new architecture is inspired from the Kolmogorov-Arnold representation theorem [4], while MLPs are based on universal approximation theorem. In brief, KANs use learnable activation functions in their training procedure instead of having static learnable weights on the network edges.

In this paper, we study the combination of KANs into the VAE architecture, proposing to replace MLPs in both the encoder and decoder components with KAN layers. By incorporating KANs with VAEs, we hypothesize that the proposed architecture will show improved performance in generative modeling, while being much more parameter efficient and stable. Moreover, because KANs decompose complex multi-variable functions with a set of simple single-variable functions while training, they can enhance interpretability considerably. The research questions we aim to investigate in this paper include:

- If we combine KANs and VAEs, how does it affect the models' generative performance? What about its ability to reconstruct input images?
- Does this integration really improve the parameter efficiency of the model compared to the traditional VAEs?
- Does this combination enhance the latent space representation?

The remainder of this paper is organized as follows: In Section II we introduce the recent efforts adapted from KAN structure. Section III & IV provide an overview of the VAEs and KANs architectures respectively. In Section V, we present the adopted methodology for our experiments. Section VI provides experimental results and comprehensive discussions regarding the performance of the KAN-VAE against the baseline VAE on various metrics. Finally, we conclude the study and propose potential directions for future research in Sections VII & VIII.

II. Related Works

Kolmogorov-Arnold Networks (KAN) [3] is one of the most recent artificial intelligence innovations inspired by both classical mathematical theory and modern deep learning. Inspired by the Kolmogorov-Arnold (KA) representation theorem [4], Liu et al. Introduced KAN as a potential alternative to Multi-Layer Perceptrons (MLPs). KA theorem states that it is possible to decompose any multi-variable continuous functions into much simpler, single-variable functions. This is super beneficial, especially in complex representation learning tasks. Since the introduction of this theorem in 1957, many attempts have been made to use KA in training neural networks [5], [6], [7]. Since most of these research efforts adopted the original KA two-layer structure, their applications in modern deep learning areas have not been fruitful. The KAN authors tried to circumvent these limitations by generalizing the KA representation theorem and introduced Kolmogorov-Arnold Networks in different widths and depths [3]. In their experiments in the original paper, they showed that this architecture has great potential in many scientific applications and can be a game changer in modern deep learning tasks.

Several papers have adapted or modified KAN architecture since Liu et al. proposed KANs [3] as a potential choice in deep learning applications. They often tried to improve the adaptability and overall efficiency of KANs in different domains. Bodner et al. Proposed Convolutional KAN [8] as an alternative to CNNs. Their experiments showed that Conv-KAN have impressive performance compared to standard CNNs while having a much smaller number of parameters. This parameter reduction is a promising direction for efficient machine learning paradigms, where researchers try to shrink the size of the neural networks so they can fit in very small hardware with limited storage and computing power. TKANs [9] and TKAT [10] are two innovative KAN modifications proposed by Genet et al. In TKAN, they integrated KANs with Long Short-Term Memory (LSTM) architecture to improve time series forecasting. While in TKAT, they combined KAN with transformer architecture to capture long-term relationships in multivariate data streams. Zhang et al. Introduced the combination of KANs and Graph Neural Networks (GNNs) [11]. They substituted MLPs and activation functions with KAN layers for feature extraction. Their research effort opens new possibilities for enhancing feature representation in graph-structured data.

Also, several papers replaced B-splines in KAN architecture with other types of functions and polynomials [12], [13], [14], [15]. To name a few, ChebyKAN [15] used Chebyshev polynomials instead of B-splines in the original KAN structure. In the FastKAN [12], Wav-KAN [14], and fKAN [13] architectures, the authors used the Radial Basis Functions (RBFs), wavelet functions, and fractional Jacobi functions respectively to improve accuracy and learning efficiency.

III. Variational Auto-Encoders (VAEs)

Variational Autoencoder (VAE) was first introduced by Kingma and Welling [1]. VAEs are a class of generative models that are inspired by Auto-Encoders (AE) and Variational Inference. Since their introduction in 2013, they have become a popular architecture in representation learning, unsupervised machine learning applications, and generative modeling [2].

Similar to the AE structure, VAE is composed of an encoder and a decoder. The encoder projects the input data to a latent space distribution and the decoder reconstructs the input by sampling from this latent distribution. The main difference between VAEs and AEs is that in VAEs, we learn the parameters of the latent distribution. While in the AEs we map input data to the points in the latent space, in VAEs we project input data to the distributions and try to learn the parameters of these distributions. This probabilistic approach enables VAEs to generate new data samples unlike in AEs which we only can reconstruct the input data [1], [2]. The VAE architecture is illustrated in Figure 1.

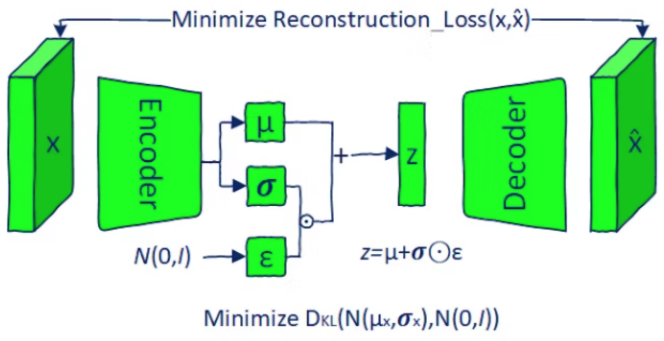

Minimize Reconstruction_Loss(x,x̂)
x
Encoder
μ
σ
ε
N(0,I)
+
z
z=μ+σ⊙ε
Decoder
x̂
Minimize D_KL(N(μx,σx),N(0,I))

Fig. 1: VAE Structure [16]

The objective function for training a VAE is as follows:

$$\mathcal{L}(x) = E_{q(z|x)}\left[\log p(x|z)\right] - \mathrm{KL}\left(q(z|x)\|p(z)\right) \qquad (1)$$

In this loss function, two terms try to optimize VAE simultaneously. The first term in the equation above is reconstruction loss. It ensures that the decoder reconstructs the input data from the latent space. The second term measures the KL divergence between the posterior and the prior. It encourages the latent distribution to be as close to the prior distribution as possible. This can be seen as a regularization term in the learning process of the VAEs. To enable backpropagation in VAE architecture, the authors used a trick called reparameterization. This trick allows the VAE model to be trained using gradient-based optimization techniques. Given all the tricks and optimization procedures discussed briefly, once the VAEs are trained appropriately, we can generate data by sampling from the latent distributions first and then passing these samples through the decoder [1].

IV. Kolmogorov-Arnold Networks (KANs)

Multi-layer perceptrons (MLPs) are fundamental components of modern deep neural architectures [17], [18]. This is because they have impressive performance in learning complex and nonlinear patterns within data. Theoretically, MLPs are inspired by the universal approximation theorem. This theorem guarantees that with appropriate activation functions and number of artificial neurons, any continuous function can be learned [17]. Despite their amazing performance in many domains such as predictive modeling, image recognition, and natural language processing, MLPs often lack interpretability (Figure 2). The huge number of connections between the layers and the explosion of model parameters make it difficult to understand the logic behind their conclusions [19]. Moreover, the MLP's often need huge computational resources such as GPUs. These limitations have motivated researchers to explore alternative architectures for MLPs.

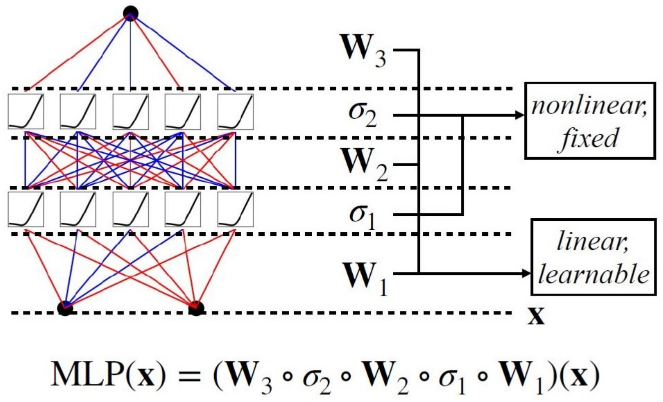

$\mathbf{W}_3$
σ_2
$\mathbf{W}_2$
σ_1
$\mathbf{W}_1$
nonlinear, fixed
linear, learnable
$\mathbf{x}$
$\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$

Fig. 2: MLP Structure [3]

One of these potential alternatives to MLPs is Kolmogorov-Arnold Networks (KANs) [3] proposed by Liu et al. (Figure 3). Unlike MLPs that use fixed activation functions at each neuron and scalar weights, KANs use learnable activation functions on the edges between nodes (neurons in KANs). These learnable activation functions are single-variable functions (usually B-spline) simply summed at each node to produce the output [3]. Theoretically, KANs are inspired by the Kolmogorov-Arnold representation theorem [4].

The KAN authors claim that their proposed architecture offers multiple benefits. First, the interpretability of the model improves significantly compared to uninterpretable MLPs. Moreover, KANs often need a fewer number of nodes (neurons) to achieve MLPs accuracy. This results in a model with a much lesser number of parameters compared to MLPs. This is significantly beneficial in efficient machine learning applications where we have limited computational resources. The experiments in the original paper suggest that KANs outperform MLPs in scientific applications, especially for modeling physical equations and solving partial differential

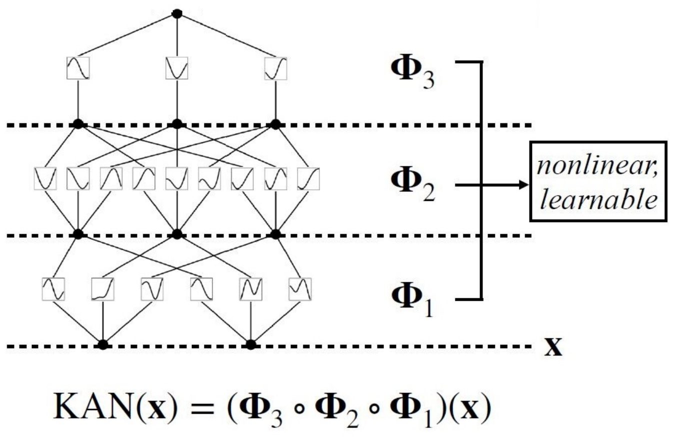

$\mathbf{\Phi}_3$
$\mathbf{\Phi}_2$
$\mathbf{\Phi}_1$
nonlinear, learnable
$\mathbf{x}$
$\text{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$

Fig. 3: KAN Structure [3]

equations. The authors also bring examples to show how KANs can be effectively used for scientific discoveries because of their improved interpretability [3].

A. KAN Representation Theorem

Kolmogorov-Arnold theorem [4] states that any continuous function with multiple variables, within a bounded area, can be expressed as a combination of a limited number of single-variable continuous functions, along with simple addition:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{2n+1} \Phi_i \left(\sum_{j=1}^{n} \psi_{ij}(x_j) \right)$$

where Φ_i and ψ_{ij} are continuous single-variable functions. This indicates that learning a complex problem with many variables can be broken down into smaller, easier-to-solve parts. However, these single-variable functions may not be smooth and differentiable in many situations. Consequently, until KANs introduction, the KA theorem had been ignored in the machine learning community for years [20], [21]. Liu et al. acknowledged the potential of applying the KA representation theorem to machine learning and scientific applications [3], motivating them to propose KAN architecture.

B. KAN Architecture Details

We can interpret KA theorem as layers of a neural network [3]. The inner functions correspond to a KAN layer where the number of inputs equals n, and the number of outputs is $2n + 1$. The outer KAN layer has $2n + 1$ inputs and a single output. Consequently, the original theorem can be seen as a 2-layer structure. Liu et al. generalized this structure to be both wider and deeper [3]. This offers several benefits. First, it gives more flexibility for learning a wider variety of functions. Second, by inspiring from the KA theorem, they introduced an architecture that's potentially more interpretable. Two things happen in each KAN layer. First, a set of learnable activation functions are applied to the input variables. Liu et al. suggested to use B-Splines as these learnable functions.

B-splines are smooth, flexible and differentiable curves. Thus, we do not face the learning challenges of the original 2-layer structure anymore. Second, all the single-variable activation functions are simply added together in each output node to produce the activation value [3]. The authors claim that this use of learnable B-spline activation functions offers a powerful structure that can solve complex problems better than MLPs. For a detailed mathematical derivations for the KAN architecture, we encourage the readers to read to the original paper. However, to understand the KAN's learning mechanism, we need to learn what are B-splines exactly.

B-Splines: B-splines are mathematical curves composed of several polynomials connected to each other. While each piece has its own shape, the whole B-spline is smooth and unbroken. They are specifically beneficial in scientific tasks such as data fitting and interpolation. B-splines are an updated version of the *Bézier* curves, offering more control over the curve's shape [22], [23], [24]. A B-spline of degree k is defined as follows:

$$\mathbf{C}(t) = \sum_{i=0}^{n} N_{i,k}(t)\mathbf{P}_i$$

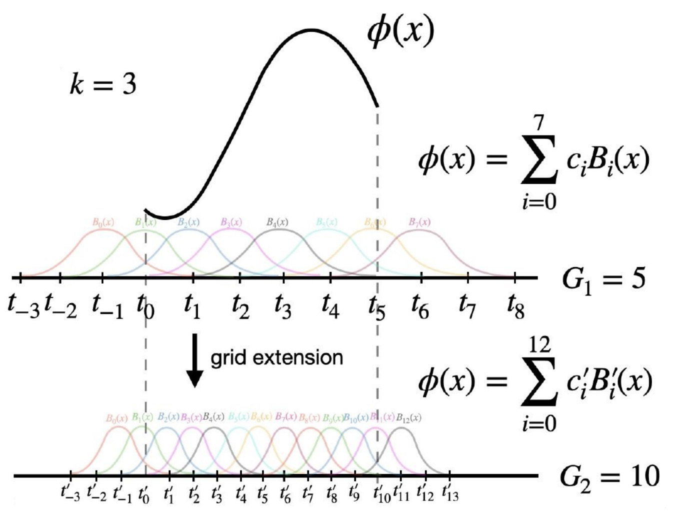
where:

- $\mathbf{P}_i$ are the control points, the points which influence the shape of the B-spline curve
- $N_{i,k}(t)$ are the basis functions of degree k
- n is the number of control points minus 1

The basis functions $N_{i,k}(t)$ are calculated using the Cox-de Boor recursion formula on the domain t which is divided into knots using a knot vector [23]. The knot vector is a list of Knot points. They define where different polynomial pieces join. The basis functions $N_{i,k}(t)$ indicate how much each control point contributes to the final curve. Thus, the final B-Spline $C(t)$ is determined by calculating these contributions. Figure 4 illustrates the learnable B-Splines as activation functions in the KAN setting.

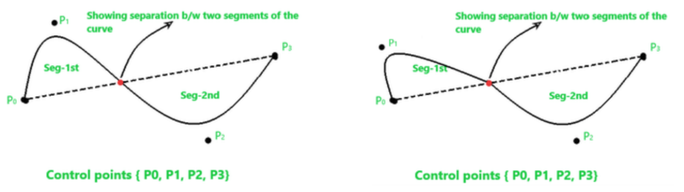
In a Bézier curve, changing the position of any control point changes the shape of the entire curve. However, in a B-spline curve, if a control point is relocated, only a specific piece of the curve reshapes. This provides more control over the curve's shape [24]. Figure 5 illustrates the effects of changing the control points on the B-spline shapes. We see that only the shape of the first segment is affected, while the shape of the second segment remains unchanged.

An interesting feature of the KAN is if we increase the number of control points (referred to as grid points G in the original paper) the expressiveness of the learnable B-spline functions will improve considerably. For example, if a network must learn a very complex multi-variable function but only has three control points (G=3), it may struggle. Instead of changing the network's structure, we can add more control points (G=10 as shown in the Figure 4) within each function. This enables the network to learn more complex relationships. This feature, named grid extension in the paper, offers a capability not found in the MLPs.



$k = 3$
$\phi(x)$
$\phi(x) = \sum_{i=0}^{7} c_i B_i(x)$
$t_{-3}\ t_{-2}\ t_{-1}\ t_0\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_8$
$G_1 = 5$
grid extension
$\phi(x) = \sum_{i=0}^{12} c'_i B'_i(x)$
$G_2 = 10$

Fig. 4: Learnable activation functions as B-Spline in KAN [3]



Showing separation b/w two segments of the curve
P1
P3
Seg-1st
Seg-2nd
P0
P2
Control points (P0, P1, P2, P3)
Showing separation b/w two segments of the curve
P1
P3
Seg-1st
Seg-2nd
P0
P2
Control points (P0, P1, P2, P3)

Fig. 5: Curve shape before (left) and after (right) changing the position of control point [24]

V. Methodology

In this study we combine Variational Autoencoders (VAEs) with Kolmogorov Arnold Networks (KANs). Specifically, we replace the typically used Multi-Layer Perceptrons (MLPs) in the encoder and decoder components of the VAEs with KANs. We hypothesize that the proposed architecture will demonstrate improved generative performance across various metrics. Our research methodology includes several aspects:

- Implementation of a KAN-VAE architecture
- Complete selection of the evaluation metrics
- Evaluating the KAN-VAE against standard VAE

In our experiments, we focus on the evaluation of generative capabilities, latent space quality, and computational efficiency of both model architectures.

A. *KAN-VAE Model Architecture*

Our proposed KAN-VAE architecture combines KANs and VAEs. It replaces MLPs in both the encoder and decoder components of the VAEs with KAN layers. The KAN-VAE architecture is as follows:

- **Encoder:** The encoder takes the input data x and projects it to the parameters of the latent distribution (μ and $log\sigma$). The whole encoding process includes three KAN layers: a) *Initial encoder*: A 1-layer KAN that maps the input to an intermediate representation. b) *Mean encoder*: A 1-layer KAN that maps the intermediate representation to the mean (μ) of the latent distribution. c) *Log-sigma encoder*: A separate 1-layer KAN that maps the intermediate representation to the log-sigma ($log\sigma$) of the latent distribution.
- **Latent Space**: The latent space is parameterized by variables μ (mean) and $log\sigma$ (log standard deviation), which are resulted from the encoder. The latent vector z is sampled from this distribution using the reparameterization trick: $z = \mu + \epsilon \odot \exp(0.5 \cdot \log\sigma), \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, I)$
- **Decoder:** A 2-layer KAN which takes in a latent vector z with the dimension of the latent space. It then tries to reconstruct the original input from this latent vector.

Similar to the standard VAE architecture, the complete forward pass of our KAN-VAE model is as follows:

1) The input x is passed through the encoder to produce μ and $log\sigma$
2) A latent vector z is sampled using the reparameterization trick
3) The sampled z is passed through the decoder to reconstruct the input

Our hypothesis is that this combination of KANs and VAEs leads to the improvement of the generative performance, parameter efficiency, and a more informative latent space.

B. Experimental Setup

1) Dataset: For our experiments, we utilized the MNIST [25] dataset. MNIST is one of the most recognized benchmarks in machine learning and computer vision. This dataset consists of 28x28 pixel grayscale images of handwritten digits ranging from 0 to 9. To simplify our experiments, we binarized the images.

2) Hardware Configuration: All experiments are conducted on Nvidia A100 GPU using Google Colab, a cloud-based platform that provides access to GPU resources.

C. Implementation Details

1) KAN-VAE: In this study, we use a modified[1] version of the KAN implementation to avoid the performance and efficiency issues of the original[2] implementation (implemented by KAN authors). This efficient implementation optimizes computational resources and improves scalability. Moreover, in our experiments, we used standard hyperparameters: 5 grid points (G = 5) and cubic splines (order k = 3). The grid points are distributed within the [-1, 1] range, without employing grid extension or sparsification techniques. We conducted manual experiments to determine optimal depths and widths for the KAN-VAE encoder and decoder components. Our findings showed that networks with more than two layers in depth yielded worse generative performance, particularly for high-dimensional input data such as images. Consequently, we chose the encoder components of our KAN-VAE to be a single-layer KAN and the decoder to be a two-layer KAN, offering superior generative quality compared to other layer structures and combinations. Additionally, we tested various widths (number of nodes) in our KAN layers through manual search. The outcomes of these experiments, including their impact on the generative performance, will be discussed in the VI section.

2) Original VAE: For a fair comparison, we implemented a standard VAE model based on an enhanced PyTorch implementation[3] of the original paper [1]. This VAE follows a general structure, designed to process the flattened 28x28 MNIST images (784 dimensions). The encoder component of the VAE consists of two fully connected layers. The first layer projects the 784-dimensional input to 400 dimensions. This is followed by two parallel layers that output the mean (μ) and log-sigma ($log\sigma$) of the latent distribution (20 dimensions). On the other hand, the decoder is composed of two consecutive linear layers. The first layer projects the 20-dimensional latent vector to 400 dimensions. The final layer reconstructs the original image. This implementation utilizes ReLU activation functions and the Adam optimizer. This is different from the original paper which used sigmoid and Adagrad.

3) Loss Function: The KAN-VAE model uses an objective function that combines reconstruction loss and Kullback-Leibler (KL) divergence, similar to the original VAE objective function (same loss functions for both architectures). The reconstruction loss measures how accurately the model reconstructs the input data. In our case with binarized MNIST data, we use Binary Cross Entropy as the reconstruction loss metric. The KL divergence term, on the other hand, acts as a regularizer for the latent distribution. It encourages that the latent distribution resembles a standard normal distribution. This improves the model's ability to generate new samples. This loss term helps both models develop a latent space that effectively learns the original data while being effective for generating new samples.

4) Training Process for both models: Both the VAE and KAN-VAE models have a same training procedure to ensure a fair comparison. Each model is trained for 30 epochs with a batch size of 32. We employed the Adam optimizer with an initial learning rate of 0.002. To adapt the learning rate during training, we implemented a ReduceLROnPlateau scheduler. This scheduler monitors the model's performance and reduces the learning rate by a factor of 0.5 when improvement plateaus. By dynamically adjusting the learning rate, the model can be optimized, especially in the later training phases when smaller updates are more useful for convergence.

[1]https://github.com/Blealtan/efficient-kan
[2]https://github.com/KindXiaoming/pykan
[3]https://github.com/pytorch/examples/tree/main/vae

D. Evaluation Metrics

1) Fréchet Inception Distance (FID): To evaluate and compare the quality of images generated by the VAE and KAN-VAE models, we used the Fréchet Inception Distance (FID) [26] score. FID is widely used in the field of generative modeling, particularly for evaluating the quality and diversity of the generated images. This score measures how similar two sets of images are by comparing the statistical properties of features extracted from the images using a pre-trained model. We used the Inception v3 [27] network pre-trained on ImageNet [28] as the feature extractor. The evaluation process is as follows:

1) We generate 20,000 images for each model evaluation
2) We perform multiple FID calculations (5 in our experiments) using different random seeds
3) We report the mean FID score

This method helps us to compare the KAN-VAE and the original VAE more accurately and reliably.

2) Number of Parameters: In addition to generative evaluation metrics, we also report the model size, quantified by the total number of trainable parameters. This metric is particularly important in the context of efficient machine learning where computational resources may be limited. The number of parameters are directly related with the amount of memory required to store the model. Models with fewer parameters typically require less computational power for both training and inference. This results in a faster execution and a lower energy consumption. In real-world situations, it's often necessary to consider both generative performance and parameter efficiency when choosing a model.

3) Visualizing Reconstruction Error & KL Divergence: To compare the performance of the KAN-VAE with the original VAE, we employed a visualization approach that plotted the components of the objective function — the reconstruction error and the KL divergence — as a function of weight updates. This plot offers insights into training dynamics, convergence speeds, and the balance between reconstruction accuracy and latent space regularization for both models. By monitoring these metrics over time, we can tell how each architecture prioritizes different components of the objective function. If KANs outperforms MLPs in this context, we expect to see the KAN-VAE achieve lower reconstruction errors while maintaining appropriate KL divergence.

4) Latent Space Visualization: Another visualization we used in our evaluation framework is latent space visualization. We used t-SNE (t-Distributed Stochastic Neighbor Embedding) algorithm to project the high-dimensional latent space into a 2D space. This visualization offers valuable insights regarding the model's capacity to learn the data representation. Compact and well-separated clusters in the latent space indicate an effective distinction between input data features.

5) Posterior Sampling: In our evaluation, we visualized posterior sampling to evaluate how well the models reconstructed and generated samples conditioned on the input images. Specifically, we selected a subset of images from our dataset and passed them through the encoder to obtain the parameters of the latent distribution. We then sampled from this distribution and used the decoder to generate new images. By comparing the original inputs with the generated samples, we can evaluate the quality of the reconstructions. This gives us insight into how well VAE and KAN-VAE models captured the data distribution in the latent space [16].

6) Prior Sampling: We also used prior sampling to evaluate the generative capabilities of our models without conditioning on specific input data. Here, we sampled random vectors from a standard normal distribution in the latent space and passed them through the decoder to generate new images. This is consistent with how VAEs are being trained, where we encourage the latent variables to follow the normal distribution. This sampling technique allows us to evaluate the models' ability to generate new samples from arbitrary points in the latent space [16].

VI. Results and Discussion

In our experiments, the standard VAE encoder has two levels of encoding. The first layer projects input features (dimension= 784) to an intermediate representation (dimension= 400). Then two fully connected layers separately project the intermediate features of the first encoding layer to the mean and standard deviation (dimension =20) of the latent distribution. The decoder is a two-layer fully connected network. The first layer samples from the latent variables (dimension =20) and maps them to an intermediate layer (dimension= 400) and the subsequent layer maps the intermediate results to the output features with a dimensionality same as the input data. It is worth mentioning that this was the original structure of the PyTorch imlpemetation of the standard VAE model for MNIST dataset. We then decided to adopt the same layer structure for our KAN-VAE implementation as well. This means that the encoder component of the KAN-VAE has two layers of encoding same as the standard VAE, but this time instead of the fully connected neural network, we use KAN layers. We ran the experiments for different widths to see how they affected the generative capabilities, stability, and parameter efficiency of the model. The results of these experiments are gathered in Table I.

A. FID Score and Parameter Efficiency

In our first KAN-VAE experiment, we chose a similar structure compared to the original VAE. This means that the KAN-VAE encoder and decoder have similar dimensions compared to standard VAE components. This model is highlighted as KAN-VAE* in Table I. We observe that choosing this huge number (intermediate dimension = 400) as the width for the KAN layer would result in the explosion of the number of model parameters. We see that for the same number of nodes (similar to neurons in MLPs) and layer structures, the KAN-VAE model has 6,512,000 parameters compared to the standard VAE which has 652,824. This explosion of parameters is because for each input feature

Model	Encoder Architecture	Decoder Architecture	FID Score	Number of Parameters
VAE	$784 \rightarrow 400 \rightarrow$ [20, 20]	$20 \rightarrow 400 \rightarrow 784$	1.22	652,824
KAN-VAE*	$784 \rightarrow 400 \rightarrow$ [20, 20]	$20 \rightarrow 400 \rightarrow 784$	1.19	6,512,000
KAN-VAE	$784 \rightarrow 50 \rightarrow$ [10, 10]	$10 \rightarrow 50 \rightarrow 784$	1.5	799,000
KAN-VAE	$784 \rightarrow 40 \rightarrow$ [20, 20]	$20 \rightarrow 40 \rightarrow 784$	1.44	651,200
KAN-VAE	$784 \rightarrow 40 \rightarrow$ [10, 10]	$10 \rightarrow 40 \rightarrow 784$	1.45	639,200
KAN-VAE**	$784 \rightarrow 35 \rightarrow$ [15, 15]	$15 \rightarrow 35 \rightarrow 784$	1.4	564,550
KAN-VAE	$784 \rightarrow 30 \rightarrow$ [20, 20]	$20 \rightarrow 30 \rightarrow 784$	1.52	488,400
KAN-VAE	$784 \rightarrow 20 \rightarrow$ [10, 10]	$10 \rightarrow 20 \rightarrow 784$	1.61	319,600
KAN-VAE	$784 \rightarrow 10 \rightarrow$ [5, 5]	$5 \rightarrow 10 \rightarrow 784$	1.63	158,300

TABLE I: FID scores and parameters count for models with different architecture structures. i.e $784 \rightarrow 400 \rightarrow [20, 20]$ is a two-step encoder. First, the input dimention 784 to intermediate dimention 400 and then from 400 to 20 as $[\mu, log\sigma]$

(pixel in our case), 400 B-spline functions are being learned in the training phase which is not sensible. One of the main reasons we wanted to use KAN is to get a more efficient model with a smaller number of parameters, but with this choice of structure we have $10\times$ more compared to the VAE model. However, we observe that this model reaches a lesser FID score (1.19) compared to the VAE model (1.22). Generally, the model with the lower FID score is performing better regarding generative capabilities. So, our KAN-VAE model slightly outperforms VAE model.

We then repeated the experiments manually for different widths (number of nodes). As we decreased the nodes in the intermediate layers, we observed that the FID score will increase, indicating that the generative capabilities decrease compared to the standard VAE. We also noticed that the number of model parameters decreased significantly as we were reducing the intermediate nodes. Among our limited experiments, the KAN-VAE model with intermediate dimension of 35 and latent variables dimensions of [15, 15] has better generative capabilities (FID = 1.4) compared to other KAN-VAE models. Although this KAN-VAE model, highlighted as KAN-VAE** in Table I, has a higher FID score compared to the standard VAE, it has a much smaller number of parameters.

NOTE: for the rest of our evaluation metrics, we only focus on VAE, KAN-VAE*, and KAN-VAE** models.

B. Reconstruction Error & KL Divergence Visualization

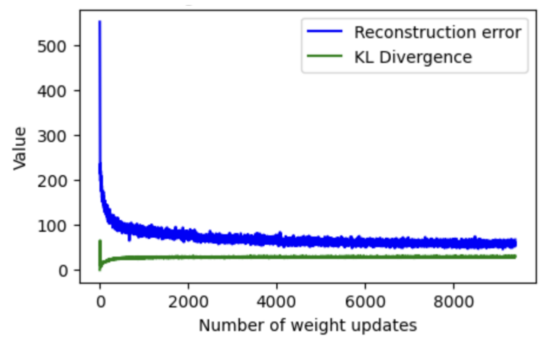
Figure 6 depicts reconstruction error and KL divergence term of the loss function in terms of the number of weight updates during training of the VAE, and both variants of KAN-VAE models. We observe that in all three sub-figures, the reconstruction error decreases rapidly in the early stages of the training and then flattens eventually. This behavior indicates that the models reach a limit where further improvement in the reconstruction of the date is not possible. All three models show similar patterns in terms of reconstruction error, with KAN-VAE* being slightly better than VAE. However, the KL divergence term shows how well the latent distribution aligns with the prior. In other words, it is an indication of the regularization in the latent space. We observe that all three models stabilize in the early phases of the training period. This demonstrates that both versions of the KAN-VAE and the standard VAE model maintain a consistent latent space. Moreover, these figures show that despite significant differences between the generative capabilities of the selected models, they have similar training dynamics. They are all stable while being trained and make a balance between reconstruction and regularization.

C. Latent Space Visualization

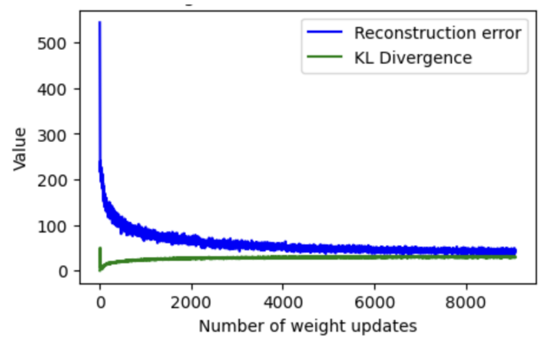
Figure 7 illustrates how the TSNE algorithm reduces the high-dimensional latent space into a two-dimensional one. Generally, if the clusters in the latent space are clearly defined and distinct from each other, it implies that the model has done a great job of encoding the input data. By looking at the three sub-figures, we can easily observe significant overlap between the clusters for the KAN-VAE**. Although in the boundary areas, the dark blue regions are well-separated, a huge number of points are scattered and overlap with others. This demonstrates that the KAN-VAE** did not do a great job of learning distinct classes of the input data and projecting it into the latent space. On the other hand, VAE and KAN-VAE* performed much better in representing different clusters, with KAN-VAE* having slightly better results compared to VAE. The clusters in these two models are well-separated and more compact, indicating their strength in learning the relationships of the input data and encoding it into latent space. There are still some minor overlaps in both sub-figures, meaning that there is still room for improvement in further differentiating between clusters.

D. Visualization of Prior and Posterior Sampling

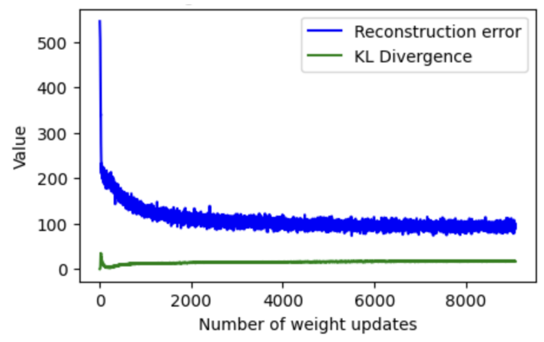
Figure 8 illustrates the posterior sampling of the three selected models. In all sub-figures, we observe that the generated posterior samples resemble the input ones. This indicates that the models excel in learning the features of the input data and reconstructing them. These observations are consistent with what we saw earlier in Figure 6 where we plotted reconstruction error throughout the training. There, we observed that the reconstruction errors dropped significantly at the beginning of the training period and then reached plateau, indicating the improvement of the reconstruction ability of the models. Although the generated samples in



(a) VAE $784 \to 400 \to [20, 20]$



(b) KAN-VAE* $784 \to 400 \to [20, 20]$

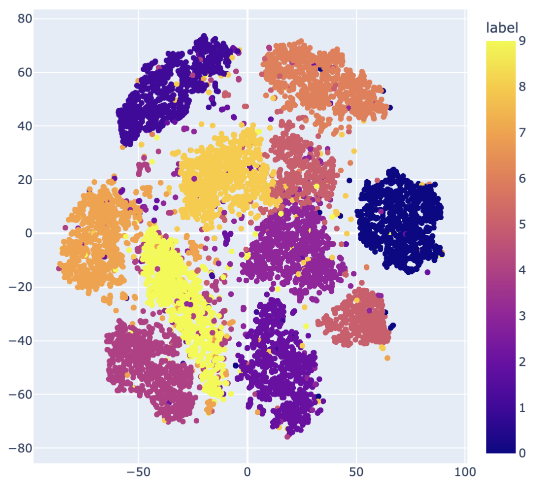


(c) KAN-VAE** $784 \to 35 \to [15, 15]$

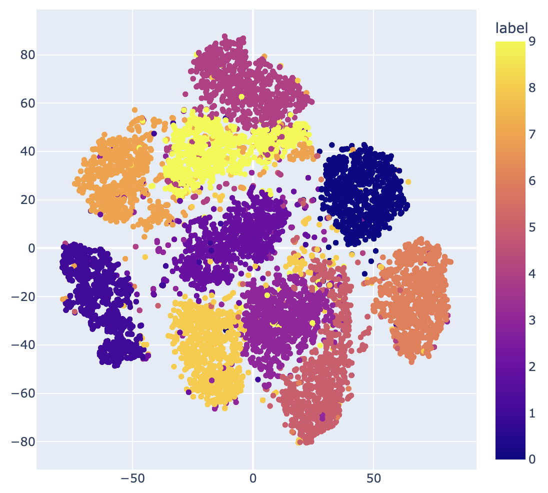
Fig. 6: Visualization of Reconstruction Error & KL Divergence

Figure 8 are a bit blurry in some cases in the VAE and KAN-VAE** models, the generated images in KAN-VAE* model are much clearer. This again is consistent with the slightly lower reconstruction error of KAN-VAE in Figure 6.

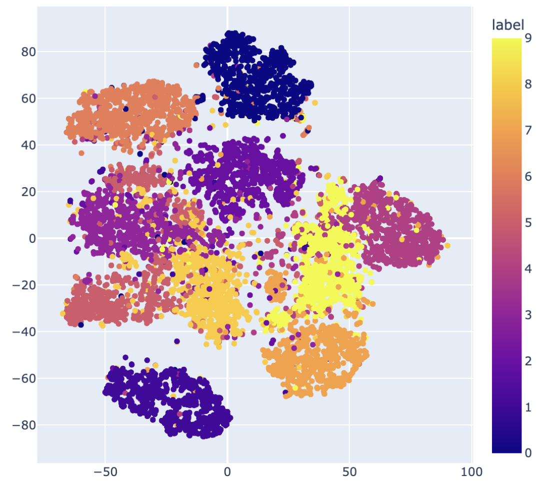
Figure 9 shows the images generated by sampling from the prior distribution without conditioning on any specific classes from the input data. The evaluation of results in these sub-figures is a bit hard since several of the generated samples are blurry and unrecognizable. However, we can observe that in the case of KAN-VAE* or even KAN-VAE**, the generated digits have more fidelity and are more recognizable compared to the VAE generated samples. This observation can indicate the potential of KAN-VAEs in generative tasks.



(a) VAE $784 \to 400 \to [20, 20]$

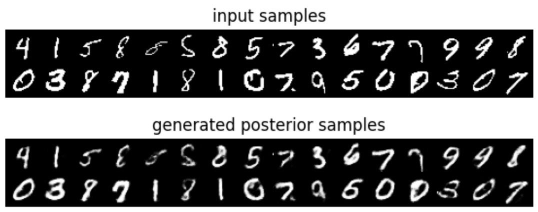


(b) KAN-VAE* $784 \to 400 \to [20, 20]$

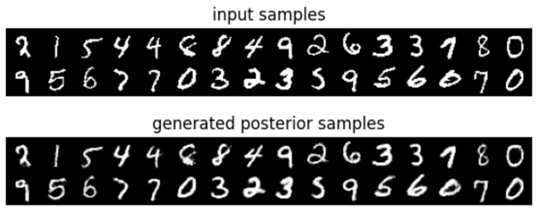


(c) KAN-VAE** $784 \to 35 \to [15, 15]$

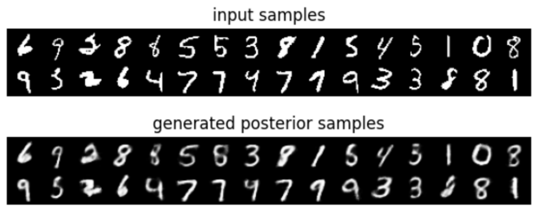
Fig. 7: Visualization of latent space with TSNE



(a) VAE $784 \rightarrow 400 \rightarrow [20, 20]$

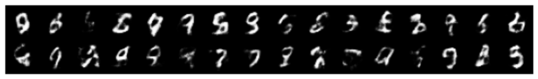


(b) KAN-VAE* $784 \rightarrow 400 \rightarrow [20, 20]$

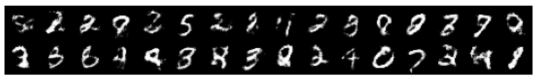


(c) KAN-VAE** $784 \rightarrow 35 \rightarrow [15, 15]$

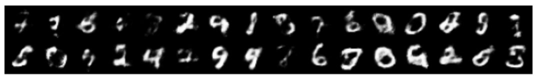
Fig. 8: Visualization of posterior sampling



(a) VAE $784 \rightarrow 400 \rightarrow [20, 20]$



(b) KAN-VAE* $784 \rightarrow 400 \rightarrow [20, 20]$



(c) KAN-VAE** $784 \rightarrow 35 \rightarrow [15, 15]$

Fig. 9: Visualization of prior sampling

VII. Conclusion

In this study, we proposed a new architecture called KAN-VAE by combining Kolmogorov Arnold Networks (KANs) and Variational Auto-Encoders (VAEs). In the beginning of the research, we hypothesized that replacing the MLP layers in the standard VAEs with KAN layers would increase the generative capability and parameter efficiency significantly. However, we observed that for having a high generative quality, we should have a large number of intermediate nodes which results in a significantly large number of parameters. One of the main reasons why we incorporated KAN into VAEs was creating a model that has better, or same generative performance compared to VAE architecture while being much more smaller and parameter efficient. One of the main reasons why our proposed architecture did not function as expected is the huge number of input features. In the original paper [3], the authors showed the strengths of this innovative architecture in small-scale AI applications and mathematical tasks i.e. PDE solving and function fitting. Consequently, their experiments involve a limited number of features for input data. For these limited input features, they demonstrated that KANs are much more interpretable and offer better accuracy. However, they did not apply KANs in big-scale Machine Learning applications, especially in generative modelling where we can have huge number of input features. Although we did not achieve the results we expected, we still think that KANs have great potential in deep learning applications and more research efforts is needed in this area.

VIII. Future Research

In our experiments, we only tested a few KAN layer structures with different widths and depths. Instead of such manual searching, we can use frameworks like neural architecture search to do it automatically for a broad number of models with different layer structures. Another interesting research path is conducting grid extension technique in the experiments to see how they affect KANs in learning the activation functions. Testing on multiple datasets with different data types could be another future direction for people who are interested in discovering the KANs potential.

References

[1] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[2] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[3] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, "Kan: Kolmogorov-arnold networks," *arXiv preprint arXiv:2404.19756*, 2024.

[4] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," in *Doklady Akademii Nauk*, vol. 114, pp. 953–956, Russian Academy of Sciences, 1957.

[5] P.-E. Leni, Y. D. Fougerolle, and F. Truchetet, "The kolmogorov spline network for image processing," in *Image Processing: Concepts, Methodologies, Tools, and Applications*, pp. 54–78, IGI Global, 2013.

[6] M. Köppen, "On the training of a kolmogorov network," in *Artificial Neural Networks—ICANN 2002: International Conference Madrid, Spain, August 28–30, 2002 Proceedings 12*, pp. 474–479, Springer, 2002.

[7] D. Fakhoury, E. Fakhoury, and H. Speleers, "Exsplinet: An interpretable and expressive spline-based neural network," *Neural Networks*, vol. 152, pp. 332–346, 2022.

[8] A. D. Bodner, A. S. Tepsich, J. N. Spolski, and S. Pourteau, "Convolutional kolmogorov-arnold networks," *arXiv preprint arXiv:2406.13155*, 2024.

[9] R. Genet and H. Inzirillo, "Tkan: Temporal kolmogorov-arnold networks," *arXiv preprint arXiv:2405.07344*, 2024.

[10] R. Genet and H. Inzirillo, "A temporal kolmogorov-arnold transformer for time series forecasting," *arXiv preprint arXiv:2406.02486*, 2024.

[11] F. Zhang and X. Zhang, "Graphkan: Enhancing feature extraction with graph kolmogorov arnold networks," *arXiv preprint arXiv:2406.13597*, 2024.

[12] Z. Li, "Kolmogorov-arnold networks are radial basis function networks," *arXiv preprint arXiv:2405.06721*, 2024.

[13] A. A. Aghaei, "fkan: Fractional kolmogorov-arnold networks with trainable jacobi basis functions," *arXiv preprint arXiv:2406.07456*, 2024.

[14] Z. Bozorgasl and H. Chen, "Wav-kan: Wavelet kolmogorov-arnold networks," *arXiv preprint arXiv:2405.12832*, 2024.

[15] S. SS, "Chebyshev polynomial-based kolmogorov-arnold networks: An efficient architecture for nonlinear function approximation," *arXiv preprint arXiv:2405.07200*, 2024.

[16] "How to sample from latent space with variational autoencoder," 2024. https://hackernoon.com/how-to-sample-from-latent-space-with-variational-autoencoder.

[17] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[18] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[19] F.-L. Fan, J. Xiong, M. Li, and G. Wang, "On interpretability of artificial neural networks: A survey," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 5, no. 6, pp. 741–760, 2021.

[20] F. Girosi and T. Poggio, "Representation properties of networks: Kolmogorov's theorem is irrelevant," *Neural Computation*, vol. 1, no. 4, pp. 465–469, 1989.

[21] T. Poggio, A. Banburski, and Q. Liao, "Theoretical issues in deep networks," *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30039–30045, 2020.

[22] "An introduction to b-spline curves," 2022. https://resources.system-analysis.cadence.com/blog/msa2022-an-introduction-to-b-spline-curves.

[23] "B-spline curve," 2009. https://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node17.html.

[24] M. Madhav, "B-spline curve in computer graphics," 2021. https://www.geeksforgeeks.org/b-spline-curve-in-computer-graphics/.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.