McGill Physical Journal

Performance and Application of Kolmogorov-Arnold Networks

Luca Monteferrante, Kai Wang, Evgeny Moiseev

McGill University Department of Physics

Abstract

Kolmogorov-Arnold Networks (KANs) is a new class of neural network that has been recently gaining attention due to its promising scaling factors and other capabilities. This project aims to understand the basic working principles of KANs and explore its application in quantum optics. The first step was learning how to build and work with KANs from a few known examples. The most pertinent of which is a performance test of the scaling laws for the loss with respect to the size of a function. In quantum state tomography, KANs were explored for fitting the measurement expectation values for reconstructing some original Wigner function describing a single-mode quantum state of light. Performance results were obtained for the loss vs parameter size, but also loss vs input size, and the observed results of KANs outperform the theoretical best that the most common neural network can obtain. Then KANs were used to approximate $\langle \hat{G}(\alpha_n, \phi_n) \rangle$ by using 1000 values of α and ϕ, within a confined domain, to get 1000 inputs of $\langle \hat{G}(\alpha_n, \phi_n) \rangle$ to the model. The KAN model was able to accurately approximate the $\langle \hat{G}(\alpha_n, \phi_n) \rangle$ with an RMSE loss of only 8.78×10^{-3} and thus it is capable of accurately recover the original Wigner function. This indicates the approximation has little to no statistical difference from the original function. These results show that KANs can be a useful tool in processing the computationally expensive data in continuous variable quantum optics.

1 Introduction

In Physics, there are often situations which are complex and difficult to analytically solve. Some examples of this include applications in astronomy with gravitational wave detection and classification [1], design of crystalline materials [2], and predicting fluid flows [3]. Such cases demand higher computational power, and one resource researchers use for help is neural networks (NN). Neural networks are described as layers of nodes, containing information of the inputted data, connected to each other via weight matrices composed of activation functions. The first layer of nodes will contain the raw information of each variable, and in each layer after it, the information will be altered non-linearly by the weight matrix and activation function [4]. When the information in the nodes is altered, it allows the network to try complex patterns with the raw data and determine whether particular operations on the inputted data are beneficial or not. To get from one layer to the next, pre-determined activation functions (operations) are the edges that link nodes from one layer to the next. They are applied onto each node in one layer and summed into each node in the next layer.

Each activation function has its own weight matrix that is trained upon [4] in hopes to realize the most intelligent neural network it can make by minimizing a loss function. The loss function determines how close the current result is to the true value. This training is what makes it machine learning (ML). The data/variable values given by the user are first split into two, one for training and one for testing. Only the training data is used for learning so that the loss is not biased on the test data. The machine uses the NN to learn what weights to apply. Over many runs/steps, the weights will change slightly and a result will be calculated, the machine will learn if this result is better or worse than the previous iteration by comparing the ground truth data supplied by the user to the result it finds. If the result with those new weights match the test data better, it will investigate this change more in future runs. The most basic and common NN are Multi-layer Perceptrons (MLP), which act exactly as described. Common uses of NN are regression tasks, similar to fitting, or classification tasks, assigning variables/points to a label.

Though these MLPs can be effective, they are not perfect, and better has been done. This imperfection becomes more and more evident as the number of variables increases and MLPs' performance deteriorates, causing a need for better NNs. Derivatives of MLPs have been made over the years, each specializing in different aspects. One such NN, proposed by Ziming Liu and other researchers [5], that has been gaining popularity recently is Kolmogorov-Arnold Network (KAN) based on the Kolmogorov-Arnold Representation theorem. The task KANs wish to improve upon from MLPs is decomposing multi-variable functions into uni-variate ones. KANs theoretically provide more accurate results than MLPs can, and in situations of fitting or regressions where MLPs don't succeed, it's possible KANs might.

In order to address the performance of KANs, there is a python library [6] for KANs, in this repository there are examples demonstrating how to use the library, as well as some examples demonstrating the performance of KANs. After research into KANs [5] and their properties is done, the examples can be ran on a local machine to test if the reported performance is truly what it seems. Then to see if KANs are useful in actual applications, a KAN model will be created for an application of quantum optics: quantum state tomography. The goal of this application being to recover the Wigner Function (a quasi-probability distribution) from the expectation value of an observable $\hat{G}(\alpha_n, \phi_n)$.

The theory of NNs (MLPs) and KANs, as well as the meaning and methods of obtaining $\langle\hat{G}(\alpha_n, \phi_n)\rangle$ measurements and recovering the Wigner function will all be explained in the upcoming methods section:2. The methods in which the data is treated will also be discussed in methods. Then a presentation of the performance of KANs, their promised capabilities, and results of the KAN application on quantum tomography will be shown in results, followed by a discussion of these findings. Finally a conclusion marking the overall success and future suggestions to further this field.

2 Materials and Methods

2.1 Neural Networks

Before explaining KANs, it's important to understand how MLPs work. As previously mentioned, there are layers of nodes connected to each other via edges (activation functions) from nodes in layer L to nodes in layer L+1. Each node x_{L+1} will be the sum of all nodes in the previous layer, with activation functions and weight matrices applied, as shown in (1). And if there are many layers, then the MLP's resulting function will look like (2) for a single variable. For a view of a simple MLP neural network, please refer to fig: 1

$$x_{L+1} = \sum_{0}^{n} (W_L \circ \sigma) x_L \tag{1}$$

$$\text{MLP}(x) = (W_{L-1} \circ \sigma \circ W_{L-2} \circ \sigma \circ \cdots \circ W_1 \circ \sigma \circ W_0)\, x \tag{2}$$

Here [5] each W represents the weight matrix applied at layer L and σ represents the activation function used. An important concept to remember is that one σ is chosen for the whole network, and cannot change unless you change it for everything else as well.

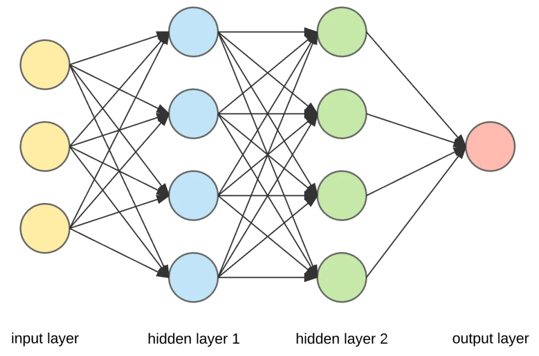

Figure 1: This is a simple MLP network. Notice how each node in the first layer is connected to each node in the next. The nodes in the first hidden layer are the summations of each node's edge (activation function and weight matrix) connected to it (1). And the 2nd hidden layer is the alteration of the hidden layer's nodes, which adds another level of complexity to the patterns the MLP can recognize, and finally the output is an alteration of those. [7]

.

Keep in mind that the weights are the objects being trained on in MLPs. To know if a weight matrix is better than another is by comparing the output of the NN to a known final result, given by the user, obtained with the same variables. This comparison is quantified by a loss function, which determines how much they differ. Both MLPs and KANs can use most loss functions, but in this paper the root mean square error (RMSE) is used (3) where the network optimizes weights for the smallest error. Any loss function could have been used, as the performance of KANs and MLPs are not specific to any loss function. A small error is a strong indicator that a network performs its task well.

$$RMSE = \sqrt{\frac{1}{N}\sum(Y - Y_{pred})^2} \tag{3}$$

where N is the number of data points, Y is the ground truth, Y_{pred} is the model's predicted values all for either the training data or the test data.

Onto KANs, the working principle behind these KAN networks is the Kolmogorov-Arnold representation theorem which states that if a function with multiple dimensions/variables exists, it can be written as series/composition of multiple simpler univariate functions [5] as shown in (4)

$$f(x) = f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right) \tag{4}$$

Here $f(x_1, \ldots, x_n)$ represents the multi-variable starting function, $\phi_{p,q} : [0,1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$ [5] where both are activation functions inside one another. This equation is proof that any multi-variable function can be decomposed into uni-variate ones.

The biggest change from MLPs to KANs are that KANs use splines as their activation functions, rather than pre-defined functions like ReLU or sigmoid. This change not only affects the need for weight matrices, but also makes the activation functions themselves learnable. In effect, the weight matrices are replaced with the splines' coefficients and each edge in the neural network is not forced to be only ReLU or only a sigmoid. [5] The splines are the objects being trained on in machine learning rather than the usual weight matrices, making them learnable activation functions. KANs convert these learnable activation functions from the current layer to the next layer with (5)

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}) \tag{5}$$

[5]. Where the letters L, i, refer to different aspects of the model. L refers to the layer number of the KAN, L = 1 means it's referring to the first layer, while $L = n_L$ is the last layer. In each L layer there are n_L nodes (neurons). Though it's also necessary to be able to see activation functions which act on the edge that links a neuron from L to L+1, so we'll call the nodes in L n_i and the nodes in L+1 n_j where i = 1, 2, ... n_L and j = 1, 2, ... n_{L+1}. Then the activation function working on the edge of neuron n_i and n_j in layer l would be written as $\phi_{l,i,j}$. The nodes before the activation function are refered to as $x_{L,i}$ and after they are called $\tilde{x}_{L,j,i}$ Each node in the next layer can be seen as a summation of activation functions from the previous layer [5] which is why linear weight matrices aren't needed in KANs. (5) can be rewritten as (6) with the same meanings for all the $\phi_{L,j,i}$. This equation can be viewed for a better visualization how each matrix of activation functions interact with each other.

$$f(x) = \sum_{i_{L-1}=1}^{n_{L-1}} \phi_{L-1,i_{L-1},j_{L-1}} \left(\sum_{i_{L-2}=1}^{n_{L-2}} \cdots \left(\sum_{i_2=1}^{n_2} \phi_{2,i_2,j_2} \left(\sum_{i_1=1}^{n_1} \phi_{1,i_1,j_1} \left(\sum_{i_0=1}^{n_0} \phi_{0,i_0,j_0}(x_{i_0}) \right) \right) \right) \cdots \right) \tag{6}$$

These changes are the only significant changes from MLPs to KANs. But why the creators of this neural network do this? It's because the math behind the accuracy of these learnable spline activation functions models is better than what MLPs can achieve. As the dimensionality of a function increases, the more difficult it is for any model to analyze/solve/decompose it. Let's define a parameter α which in a plot shows the slope of a model's root mean square loss (RMSE) vs the number of dimensions it has. This slope is relevant as it represents the likelihood that expanding the model, either in terms of increasing layers or increasing number of neurons in each layer, would result in more accurate function decompositions. The steeper the slope α, the greater the chance of improvement. Theoretically MLPs should have an α which scales like $-(k+1)/d$ [5] where k is the polynomial used (k is 1 in ReLU) and d is the dimensionality of the function being output. So the larger the dimensionality is, the worse the approximation becomes. But this is not so much of an issue for KANs because their goal is to decompose complex functions into uni-variate functions, meaning d = 1. And for KANs, cubic splines are most commonly used, implying k = 3, this increasing the slope of α in comparison to that of MLPS. Then if KANs act in practice as they should in theory, they should be more accurate than MLPs as the dimensionality increases. Though due to all these changes in learnable activation functions and reducing dimensionality to 1, this causes KANs to learn slower than MLPs do.

2.2 Quantum Tomography

The specific application which KANs are being applied to is quantum state tomography. This experiment has some unknown pure quantum state described by a Wigner function W(x,p), in hopes to be able to recover that Wigner function (7) from multiple measured observables. W(x,p) is a quasi-probability distribution describing the position and momentum operators (x,p) in phase space of an electromagnetic wave. Quasi-probability means that the distribution can have negative probabilities, as well as the distribution not needing to sum to 1 over all space. Using measured observables to recover quantum states is exactly what quantum tomography represents. It can be thought of as the opposite of quantum mechanics operations; an example of these operations could be finding the wavefunction, then using that to generate expectation values. In quantum tomography the expectation values would instead be used to find the wavefunctions.

$$W(x,p) = \frac{1}{\pi\hbar}\int_{-\infty}^{\infty}\psi^{\star}(x+y)\psi(x-y)e^{-2ipy/\hbar}dy = \frac{1}{\pi\hbar}e^{-((\frac{x-\langle x\rangle}{\sigma})^2+(\frac{\sigma(p-\langle p\rangle)}{\hbar})^2)} \tag{7}$$

Where x and p are the position and momentum values in phase space and y is a dummy variable for position, and $\psi(x)$ is a gaussian wavefuntion. This representation is true for a pure quantum state.

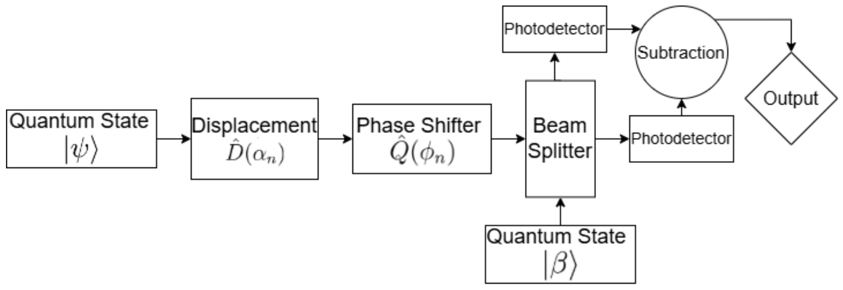
Quantum State $|\psi\rangle$
Displacement $\hat{D}(\alpha_n)$
Phase Shifter $\hat{Q}(\phi_n)$
Beam Splitter
Quantum State $|\beta\rangle$
Photodetector
Photodetector
Subtraction
Output

Figure 2: $|\psi\rangle$ is the unknown state described by a wigner function, $|\beta\rangle$ is a coherent state, $\hat{D}(\alpha_n)$ is the displacement operator, $\hat{Q}(\phi_n)$ is the rotation operator which applies a rotation of ϕ_n. The beam splitter introduces interference with the unknown state (after the displacement and phase shift) and the coherent states, which are split into two detectors which are subtracted from each other to get $\hat{G}(\alpha_n, \phi_n)$.

In order to recover a Wigner function, the setup in fig:2 is used to obtain $\langle\hat{G}(\alpha_n, \phi_n)\rangle$. This setup represents a homodyne detection setup, which means that the quantum state $|\psi\rangle$ is being interfered by a local oscillator $|\beta\rangle$. This setup also makes use of balanced photo-detection (which means it uses two photodetectors) to capture two versions of the signal. When they are combined in the output, the amplitudes will be subtracted and a measurement obtained. Using this setup once obtains one such measurement and multiple uses provide access to the wigner function at different positions of the phase space with different $\langle \hat{x} \rangle$ and $\langle \hat{p} \rangle$. These can then be used to obtain $\hat{G}(\alpha_n, \phi_n)$, which is the measurement operator associated with this setup, as shown in the results section. Acquiring multiple observables allows for a lot more data stored to extrapolate characteristics of the Wigner function. The $\langle \hat{G}(\alpha_n, \phi_n) \rangle$ function that is obtained will be used to recover this Wigner function used to make it. Firstly, $\hat{G}(\alpha_n, \phi_n)$ is defined as $\hat{D}^{\dagger}(\alpha_n)\hat{Q}(\phi_n)\hat{D}(\alpha_n)$, which can be plugged into (8) and then into (9) to obtain $\langle \hat{G}(\alpha_n, \phi_n) \rangle$.

$$g_n(x, p, \alpha_n, \phi_n) = \frac{1}{\pi\hbar} \int_{-\infty}^{\infty} \langle x - y | \hat{G}(x, p, \alpha_n, \phi_n) | x + y \rangle e^{-2ipy/\hbar} dy \tag{8}$$

$$\langle \hat{G}(\alpha_n, \phi_n) \rangle = \int_{-\infty}^{\infty} dx dp W(x, p) g_n(x, p, \alpha_n, \phi_n) \tag{9}$$

$\langle \hat{G}(\alpha_n, \phi_n) \rangle$ will be used as the ground truth data for KANs, as the goal of this application is to see if KANs can be used to recover the entire $\langle \hat{G}(\alpha_n, \phi_n) \rangle$ given a set of measurements outcomes of that observable. Then using the Husimi Q function, which represents the quasi-probability density of the unknown state overlapping the coherent state. Where for the coherent state β Q is represented as (10) which can be used to recover the wigner function in a change of variables as in (11)

$$Q(\beta) = \frac{1}{\pi} \langle \beta | \hat{\rho} | \beta \rangle = \frac{1}{\pi} e^{-|\beta - A|^2} \tag{10}$$

$$W(\gamma) = \frac{2}{\pi} \int_{\infty}^{\infty} d\beta^2 Q(\beta) e^{-2|\gamma - \beta|^2} \tag{11}$$

Where A is a complex amplitude in phase-space specific to the Wigner function that's attempted to be recovered, $\rho = |A\rangle\langle A|$, γ is a dummy variable for a general complex amplitude in phase-space. But A and γ should be scaled in the same manner.

3 Results

The results will be split into two sections, the first being example 3, the most important example, from the GitHub repo [6], the second being the results of the fitting on $\langle \hat{G}(\alpha_n, \phi_n) \rangle$ and the resulting Wigner function.

3.1 Example 3

Example 3 goes over training a KAN model on a function and tries to estimate the relationships between the parameters and replicate the ground truth formula: $\exp(\sin(x_1^2 + x_2^2) + \sin(x_3^2 + x_4^2))$. 3000 values for each variable are created, and tuples of 4 are inputted into the formula to get a label that KAN trains on. The model will try to recreate a formula that best approximate the 3000 values only knowing those 3000 inputted tuples. This formula would imply a starting layer with 4 nodes since there are 4 variables. Then using a loss function the resulting predicted formula was compared to the ground truth. The model was trained with grid amounts equal to [3,5,10,20,50] which would change after each model reaches 50 steps. In the pykan library if there are n variables then it will create a n-dimensional space containing the entire domain of each dimension. The grid argument can be passed to pykan to allow for grids. What the grids do is split the n-D space into subsections. If grid = 3 then the entire n-D space will be split into 3 grids. Each grid has its own spline that is trained inside it, and therefore its own set of parameters. If a tuple's value falls withing the domain of one of the grids, then that grid's spline will be trained on that tuple. The grid size is changed for each x-value in the fig3. This, as well as the shape of the KAN model being trained, need to be taken into account when calculating the number of parameters used in the x-axis of fig:3. As the grid value increases, so do the parameters that need to be learned. This relationship is $numparams = (|N| + |E|) * grid$ where $|N|$ is the number of nodes not in the first layer and $|E|$ is the number of activation functions. This example is to demonstrate the scaling potential of the KAN's accuracy vs number of parameters and how they relate to the theoretical scaling limit. The observed scaling does not follow the theoretical limit on my machine in fig:3 and neither does it on the official repo's [6] version fig:8. But regardless of it's comparison to the theoretical limit, they both still outperformed even the best a MLP could theoretically do.

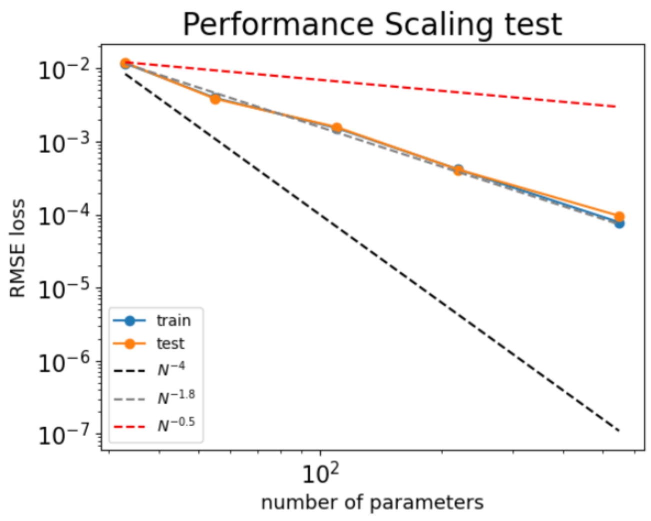

Figure 3: The blue and orange lines represent the train and test RMSE. The black dotted line represents the theoretical scaling limit of KANs ($\alpha = -4$) and the grey dotted line represents the approximate scaling limit of the training and test results ($\alpha = -1.8$). The red line represents the theoretical scaling limit of MLPs $\alpha = -0.5$

Both the obtained losses and the GitHub's losses were rounded to 3 decimal places. The observed RMSE test losses were [0.0120, 0.00387, 0.00158, 0.000413, 9.62e-05] in comparison to GitHub's RMSE test losses of [0.00923, 0.00325, 0.00107, 0.000375, 3.78e-05] each for this number of parameters: [33, 55, 110, 220, 550].

3.2 Quantum Tomography Results

For this application, The KAN model needs ground truth data of $\langle \hat{G}(\alpha_n, \phi_n) \rangle$. Starting from the gaussian Wigner function (7), one can use (8) and (9) to obtain the following expectation value (12)

$$\langle \hat{G}(\alpha_n, \phi_n) \rangle = cos(\phi_n)(\langle \hat{x} \rangle + \Re(\alpha_n)) + sin(\phi_n)(\langle \hat{p} \rangle + \Im(\alpha_n)) \tag{12}$$

Where for this experiment, let $\langle \hat{x} \rangle = 1$ and $\langle \hat{p} \rangle = 0.5$. For the KAN to train on this function, multiple inputs are required. The following domains were created for random sampling on three variables ϕ_n, $\Re(\alpha_n)$, $\Im(\alpha_n)$ are [0-2π], [0-3], [0-3]. The architecture of the KAN model used is portrayed in fig:4 with grid size = 8, spline being cubic splines (k=3), and 1000 training inputs and 200 test inputs.

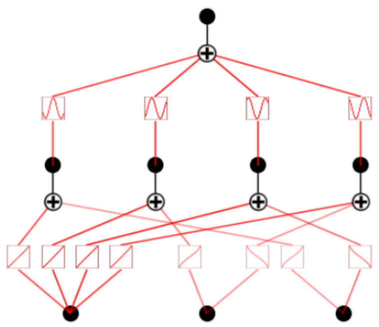

Figure 4: The bottom nodes represents the variables. From left to right they are ϕ_n, $\Re(\alpha_n)$, $\Im(\alpha_n)$. This model was trained with the LBFGS optimizer for 200 runs, grid size = 8, k = 3, with 1000 input functions for training and 200 for testing.

The choice of 1000 training inputs was made from the results of fig:5. As the training size increases, overall the accuracy of the model should increase, but not always with steady improvement in NN models.

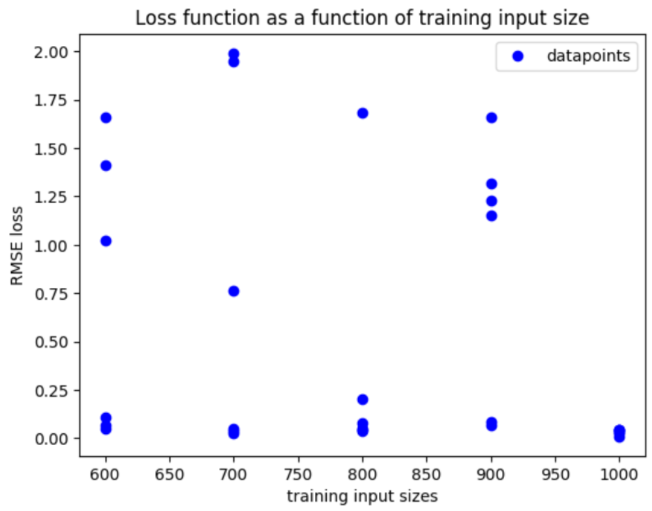

Figure 5: This plot contains 6 different simulations. The only training input size that converges to a loss of 0 is when the training input size is 1000. This implies that 1000 datapoints are enough to eliminate most randomness from the goodness of the fit.

As discussed in the intro and methods, KANs purpose is to decompose multivariable functions into compositions of univariate functions. With a KAN model defined as fig:4 the following prediction for $\langle \hat{G}(\alpha_n, \phi_n) \rangle$ was made (13).

$$\langle \hat{G}(\alpha_n, \phi_n) \rangle_{pred} = -7.893 \sin(0.9994\phi_n + 0.06237\Im(\alpha_n) + 1.710) + 8.538 \sin(0.9995\phi_n - 0.06080\Im(\alpha_n) + 7.726) - 9.316 \cos(0.9997\phi_n - 0.05646\Re(\alpha_n) + 4.693) + 7.646 \cos(0.9999\phi_n + 0.06248\Re(\alpha_n) - 1.548) - 0.001283 \tag{13}$$

A visual comparison for each test input of the predicted expectation value (13) vs ground truth value (12) gives the following figure 6 with its residuals in fig:7.

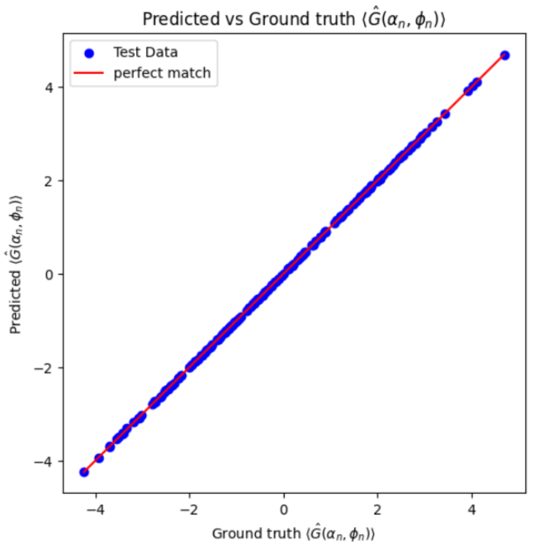

Figure 6: The x axis represents the ground truth expectation values while the y axis represents the values that the KAN model predicts it to be. If they are the same then they will be along the red line of y=x

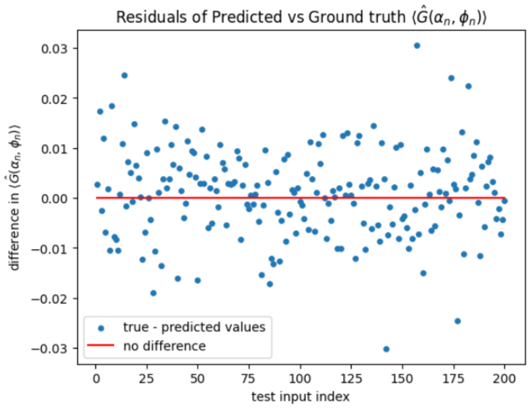

Figure 7: This graph represents the residuals of fig:6. The x axis represents the number of the test that's being compared. So x = 1 compares the true values for a specific ϕ_n and α_n with the predicted values from the KAN model. The y axis represents the difference in the value, specifically the true value subtracted by the predicted value.

Now it's time to see if it's possible to recover W(x,p) from $\langle \hat{G}(\alpha_n, \phi_n) \rangle$. Note that if $\langle \hat{G}(\alpha_n, \phi_n) \rangle_{pred}$ takes the same form, then the operations to recover W will be identical and trivial, thus the importance of the fitting is such that it does resemble an identical form such that it can recover the correct Wigner function. To extract the A value intrinsic to the Wigner function, it's necessary to average over ϕ_n in $\langle \hat{G}(\alpha_n, \phi_n) \rangle$. This is done to remove the displaced amplitudes from the measurements and obtain the clearest versions of the expected position $\langle \hat{x} \rangle$ and expected momentum $\langle \hat{p} \rangle$, which results in

$$\langle \hat{G}(\phi_n) \rangle = \langle \hat{x} \rangle \cos(\phi_n) + \langle \hat{p} \rangle \sin(\phi_n) = \cos(\phi_n) + 0.5 \sin(\phi_n) \tag{14}$$

From this equation, we can see that the complex amplitude $A = \langle \hat{x} \rangle + i \langle \hat{p} \rangle$, but can be rescaled to $A = \frac{\langle \hat{x} \rangle}{\sqrt{2}\sigma} + \frac{i \langle \hat{p} \rangle \sigma}{\sqrt{2}\hbar}$ and thus $\gamma = \frac{x}{\sqrt{2}\sigma} + \frac{ip\sigma}{\sqrt{2}\hbar}$. After plugging in these values into (10) and then (11), The following results are made (15)

$$W(\gamma) = \frac{2}{\pi} e^{-2|\gamma - A|^2} \Rightarrow W(x,p) = \frac{1}{\pi\hbar} e^{-((\frac{x - \langle x \rangle}{\sigma})^2 + (\frac{\sigma(p - \langle p \rangle)}{\hbar})^2)} \tag{15}$$

The change in coefficient in front of the exponential from $\frac{2}{\pi}$ to $\frac{1}{\pi\hbar}$ is due to the Jacobian of the change of variables. When you change γ And A back into phase space coordinates, a Jacobian of $\frac{1}{2\pi}$ is multiplied. After all is done, the correct Wigner Function is obtained.

But can the same be said for $\langle \hat{G}(\alpha_n, \phi_n) \rangle_{pred}$? When (13) is averaged over ϕ_n, and simplified, the following formula is obtained (16)

$$\langle \hat{G}(\phi_n) \rangle_{pred} = 1.002cos(0.999\phi_n) + 0.514sin(0.999\phi_n) - 0.00128 \tag{16}$$

Such a small value of -0.00128 can be neglected, due to its insignificant magnitude and lack of effect on the trigonometric functions which define the expectation value. The form that KAN spits out is identical to that of the ground truth, just the value of $\langle \hat{p} \rangle$ and $\langle \hat{x} \rangle$ are slightly different. $\langle \hat{x} \rangle$ is 1.002 instead of 1, $\langle \hat{p} \rangle$ should be 0.5 but was predicted to be 0.514. The difference in coefficient of ϕ_n from 1 to 0.999 is irrelevant. Though the values are very similar, it's important to measure how different they actually are to each other. For consistency a loss function of RMSE will be used to quantify the difference. This value when comparing (14) with (16) results in RMSE = 8.78×10^{-3}. For values ranging up to values of 4, that is quite small. Calculating the Wigner function results in the same equation as the ground truth 15 but with $\langle \hat{x} \rangle$ = 1.002 and $\langle \hat{p} \rangle$ = 0.512. Thus resulting in an accurate reconstruction of the original Wigner function.

4 Discussion

4.1 Example 3

This initially did not reproduce the outputs that the GitHub creator had. Initially the training and test losses that we obtained were much greater than those in the GitHub, and the slope of the losses were accordingly much worse. The fix which resulted in getting losses that were all within the same magnitude of those in the GitHub, is allowing the gradients of the backpropagation to change. Not allowing them to change was not of my doing but an error of the pykan library which had to be found and fixed. When it started to work, the scaling factor began acting like it had an α of -1.8. This scales better than how a MLP would scale since $\alpha = -(k+1)/d$ and for MLPs and in this example, d = 4 and k = 1. So, α would be -0.5 in the best case of MLPs. This suggests that even in the theoretical perfect performance for MLPs, KANs can outperform them even when they are not performing optimally. This result is important as it demonstrates that KANs are definitely more reliable than MLPs, which imply that in future situations which may involve more complex formuli, KANs should definitely be trusted to be more effective than MLPs.

The reason pykan could not be performing optimally can be due to the training of the KAN model. While following the logic of the GitHub creator, and fixing any of their mistakes, it became clear they wanted to simply show how KANs acted and not obtain optimal results. To get results closer to the optimal scaling, allowing the models to train for more runs could improve the quality of the approximations and reduce the RMSE loss. This is possible because the model was still improving for most of the parameter amounts that were tested. Training should only stop when overfitting occurs (which is when the test is much worse than the training) or when the improvements cease, which neither happened. But regardless The Github's logic was followed as to not differ from their epochs and have incomparable results to theirs. The GitHub's version can be found in fig:8. It's also a shame that the creator did not include grid sizes large enough to find a number of parameters which cause KANs enough difficulty to train that it stops increasing. It would have been educative to have this point when the slope α becomes 0, such that it's known that the model can no longer improve it's results.

4.2 Quantum tomography

Many different combinations of amounts of hidden layers, width of hidden layers, grid sizes, amount of runs, and pruning amounts, and also different amount of training data were attempted. The combination that resulted in the lowest loss and thus the smallest difference between the actual $\langle\hat{G}(\alpha_n, \phi_n)\rangle$ is detailed in the caption of 4. The reason this architecture makes sense is because there are 3 variables which can change. Of course ϕ_n is a real number but α_n is a complex number such that $\alpha = a + ib$ where a and b are both variables that change. This is the reason the initial layer contains 3 nodes. There is only 1 hidden layer because in the ground truth (12), there are only two functions: cosine and a sine which are added to each other. This means KANs only need one hidden layer. If for insstance there was a sine inside a cosine, then it's likely that the KAN would require 2 hidden layers. After attempting multiple widths for this layer, a width of 4 is best likely due to there being 4 terms. If you expand (12) this can be clearly seen. And finally the output layer is only 1 node.

The inner workings of NN training are generally understood but the specifics per model can't be, though behavior like shown in fig:5 is common. This is due to concepts such as nonconvex optimization, random sampling. Nonconvex optimization refers to the 3-D domain space created by the model. Convex spaces are best for NN as there is one deepest part of the space which represents the lowest loss. All NNs aim to reach this deepest point. But in spaces with many deep points, many flat regions, or anything similar, it's not always possible for the model to reach the best point with the best loss. The way the model traverses this space is by training, training changes the location of the point and the NN is attempting to reach the deepest point. This optimization is called gradient descent. NNs are not always optimal because it's dependent on the training inputs. These training inputs work as a starting point for the model to learn. It can be much easier or much harder, to reach the optimal loss depending on the starting point. More inputted data points to train on does not necessarily mean that the data points are in advantageous positions. It's possible (but increasingly unlikely as the amount increases) that the opposite is true, which would lead to the non-linear patterns shown in the figure. Though at 1000 data points it seems to converge to a very accurate value.

In all attempts, the runs with 1000 training inputs yielded the best results (loss of $8.78 * 10^{-3}$). In regression tasks, it's difficult to find a suitable formula for accuracy, which is why loss functions are usually used to quantify the difference and/or distance of two functions. A loss of 3 magnitudes smaller than the observed and expected values is a strong indication that the approximation function obtained by KAN is very accurate. Other evidence to suggest that the approximation is accurate is the residual plot from fig:7. It depicts a random assortment of minuscule errors of maximum magnitude $3 * 10^{-2}$. The randomness of the shape along with the small amplitude is another strong indicator of an accurate prediction of a function.

All this evidence leads one to believe that KANs have successfully approximated $\langle\hat{G}(\alpha_n, \phi_n)\rangle$ and thus can accurately recover the Wigner function W(x,p) from its measured expectation value.

5 Conclusion

With the results of the scaling laws and fitting of the Kolmogorov-Arnold Networks to recover a Wigner function, it's safe to say KANs are a promising neural network for regression and function decomposition tasks. It scales better with respect to parameter size than MLPs can and is accurately able to fit an expectation value of an observable such that the difference between $\langle\hat{G}(\alpha_n, \phi_n)\rangle_{pred}$ and the true $\langle\hat{G}(\alpha_n, \phi_n)\rangle$ is a magnitude of 3 less than the true value. KANs can be seen as a clay model, to smoothen out rough patches while MLPs are blocky constructions which can overall model well, but not well enough for high complexity models. KANs have proven themselves able to outcompete MLPs, thus KANs should be further applied on larger applications with more parameters. In order to not only further test the limitations of KANs but more importantly, to have a system that can bring more accurate results in other physics experiments.

References

[1] D. George and E. A. Huerta, "Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced ligo data," *Physics Letters B*, vol. 778, pp. 64–70, 2018. 1

[2] T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties," *Physical review letters*, vol. 120, no. 14, p. 145301, 2018. 1

[3] J. Ling and J. Templeton, "Evaluation of machine learning algorithms for prediction of regions of high reynolds averaged navier stokes uncertainty," *Physics of Fluids*, vol. 27, no. 8, 2015. 1

[4] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994. 1

[5] 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. 1, 2, 3, 4

[6] K. Xiaoming, "pykan: A python library for kanban board management," https://github.com/KindXiaoming/pykan, 2025. 1, 5, 6, 12

[7] A. Vidhya, "Introduction to the neural network model, glossary, and backpropagation," 2022, accessed: April 14, 2025. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/01/introduction-to-the-neural-network-model-glossary-and-backpropagation/ 2

A GitHub's version of Example 3

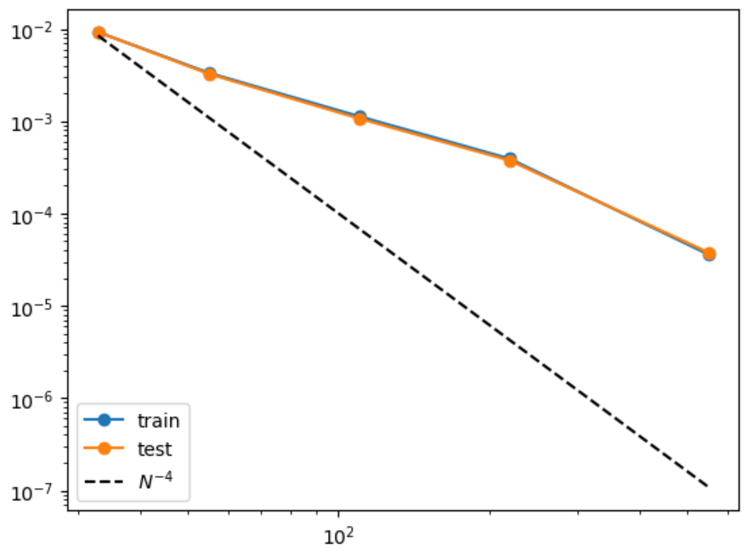

Figure 8: This image is pulled from the GitHub [6], in theirs they did not label the axes or title the graph. The blue and orange lines represent the train and test RMSE. The black dotted line represents the theoretical scaling limit of KANs ($\alpha = -4$).