## Kolmogorov-Arnold Networks:

## Implementation and Comparison with MLPs

### Abstract

Kolmogorov-Arnold Networks (KANs) is an alternative neural network architecture inspired by the Kolmogorov-Arnold Representation Theorem. KANs have learnable activation functions on edges (“weights”) whereas traditional Multi-Layer Perceptrons (MLPs) have fixed activation functions on nodes (“neurons”). This report examines the theoretical foundations of KANs, the implementation details by using the MNIST dataset, and the comparison of MLPs and KANs performance in terms of loss values and accuracy.

### Introduction

Multi-Layer Perceptrons (MLPs), which take significant importance on the deep learning task have power to approximate the nonlinear functions. This power comes from the Universal Approximation Theorem. However, MLPs rely on fixed activation functions applied uniformly across all neurons, limiting their flexibility and interpretability. The KANs are offered as an alternative to MLPs by introducing learnable activation functions on edges which make the model more interpretable and accurate. Consequently, at the end of the day, KANs have no linear matrices since each weight parameter is replaced by a learnable univariate function parametrized as splines and KANs’ nodes simply does summations (‘this causes a smaller computational graph than MLPs’). This architectural change allows KANs to dynamically adjust activation functions during training, making them more adaptable to complex data patterns.

As mentioned, KANs are designed to combine the strengths of splines and traditional neural networks. Splines are valid in low-dimensional problems and approximating univariate functions due to their accuracy and smoothness. However, they have a huge curse of dimensionality (COD) problem. In other words, as the number of inputs increases, the number of parameters increases exponentially. On the other hand, MLPs are better at capturing high-dimensional compositional structures by suffering from COD. A function can be approximated by the model which learns compositional structure well and approximate univariate functions. KANs are the models that have MLP structures on the outside and splines on the inside. By this, KANs can effectively learn features and optimize these learned features with great accuracy.

### Theoretical Background

The Kolmogorov-Arnold Representation Theorem states that any continuous multivariate function can be represented as a composition of continuous univariate functions and summation operations. KANs utilize this theorem by replacing traditional weight parameters in MLPs with learnable activation functions parameterized as B-splines. These splines allow for fine control over local regions of the input space, making the network more precise in approximating complex functions.

### Implementation Details

* **Dataset:** MNIST (28x28 grayscale images)
* **Models:** SimpleKAN and SimpleMLP with AdamW Optimizer
* **Frameworks:** JAX, Flax, TensorFlow
* **Optimization Algorithm:** AdamW with adjusted learning rate and ‘0.0001’ weight decay
* **Learning Rates:**
  + KAN: Scheduled learning rate starting at 3e-5
  + MLP: Fixed learning rate of 1e-4
* **Epochs:** 4
* **Batch Size:** 128

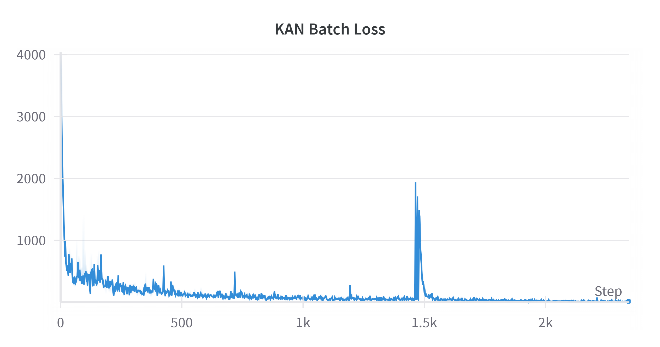
The MNIST dataset, a standard benchmark for image classification tasks, consists of handwritten digits from 0 to 9. The images were preprocessed and normalized before feeding them into both models. The SimpleKAN model was implemented with spline-based activation functions, while the SimpleMLP model used traditional fixed activation functions “Relu”. Additionally, a dropout layer is introduced on both models to prevent overfitting.

### Experimental Results

The experiment compared the performance of KANs and MLPs on the MNIST dataset. Evaluation metrics included training loss and test accuracy.

A graph with blue lines

Description automatically generatedA graph with a line

Description automatically generatedA graph with a red line

Description automatically generated

Figure 1.2 MLP Batch Loss

Figure 2.2 MLP Accuracy

Figure 1.1 KAN Batch Loss

Figure 2.1 KAN Accuracy

The KAN batch loss graph shows a fast initial drop in loss, showing quick learning at the start. However, there are sudden spikes, possibly caused by unstable gradient updates. Despite this, the overall loss decreases and stabilizes. On the other hand, the MLP batch loss graph shows a slower but steady decrease with more consistent noise throughout training. In terms of accuracy, KAN initially improves rapidly and reaches about 89%, but slightly drops at the end, likely due to overfitting or an overly aggressive learning rate adjustment. MLP accuracy improves gradually and stabilizes at around 85%, showing slower but consistent learning. The overall loss graph for KAN has a clear downward trend but shows a slight rise at the end, again suggesting sensitivity to hyperparameter settings or fine-tuning issues.A graph with a red line

Description automatically generatedA graph with a line going down

Description automatically generated

Figure 3.1 KAN Loss

Figure 3.2 MLP Loss

### Conclusion

The experimental results show that KANs learn faster and achieve higher accuracy than MLPs. However, they are more sensitive to learning rate adjustments, which sometimes cause instability or spikes in batch loss. Dropout layers helped both models reduce overfitting, leading to better generalization. While KANs converge faster, they require careful hyperparameter tuning to avoid issues like sudden spikes or slight accuracy drops.

As a result, Kolmogorov-Arnold Networks present a promising evolution in neural network design by leveraging learnable edge-based activation functions inspired by the Kolmogorov-Arnold Representation Theorem. Experimental results demonstrate their superiority over traditional MLPs in terms of accuracy, convergence speed, and interpretability.

### Future Work

* Apply KANs to more complex datasets, such as CIFAR-10 or ImageNet.
* Optimize training efficiency and reduce computational overhead.
* Investigate interpretability enhancements for KAN models.
* Refine hyperparameters to avoid overfitting.

### References

* Liu, Z., et al. (2024). KAN: Kolmogorov–Arnold Networks.