A Benchmarking of Kolmorogorov-Arnold Transformers with Taylor Polynomial Base Function on Seq2Seq Syntactic Transformation Tasks

Vincent Li
Yale University
vincent.li.vl298@yale.edu

Jared Wyetzner
Yale University
jared.wyetzner@yale.edu

Tyler Chen
Yale University
tyler.chen.tzc2@yale.edu

Abstract

The Kolmogorov-Arnold representation theorem states that any multivariate continuous function can be represented as a sum of continuous single-variable functions. We seek to leverage the theorem to design a new transformer architecture, the Kolmogorov-Arnold transformer, that is more memory-efficient than the traditional transformer while also retaining its desirable level of performance on natural language tasks, such as English question formation. In particular, we propose a novel kind of Kolmogorov-Arnold transformer that employs a Taylor polynomial as its base function, in contrast to the rational base function that has been explored in previous works. We then benchmark our architecture against previous Kolmogorov-Arnold architectures and other neural network architectures, including LSTMs, transformers, and tree transformers, finding that the polynomial transformer encounters the exploding gradient problem and does not complete training, while the rational function Kolmogorov-Arnold transformer learns the patterns well in the train and test set but does not generalize to unseen grammatical structures well. We analyze the results and give a few recommendations and suggestions for promising areas of future work based on this paper.

1 Introduction

Natural language processing is a wide and varied field, and there exist many learnable natural language tasks and computational approaches towards solving them. In this paper, we focus on English question formation. As an example, the declarative sentence

the walrus doesn't read

would be transformed into the question

doesn't the walrus read ?

In the dataset we used (McCoy et al., 2020), there are also examples in which the sentence is given in declarative form and not required to be transformed at all. Given that correctly transforming a declarative sentence into a question in all cases requires knowledge of the language, we consider high performance on this task to be representative of the model having acquired some generalizable knowledge of the English language. This is because, in order to properly learn the transformation, the model must learn the hierarchical rule for syntactic transformation, and in doing so, must learn about the tree-like structure of syntax in English, which confers general knowledge of the syntax of the language itself to the model.

In this paper, we seek to introduce a new Kolmogorov-Arnold network (KAN) architecture and benchmark it against existing approaches, which include LSTMs, transformers, tree transformers, and previous KAN architectures.[1]

1.1 The Tree Structure of English

In the question formation task that we use, McCoy et al. (2020) note that, in the training data, the rule for transforming a sentence into a question is consistent with two transformation rules: a linear rule (also referred to as *move-first* by McCoy et al. (2020)) and a hierarchical rule (also referred to as *move-main* by (McCoy et al., 2020)). Although the details of both rules are beyond the scope of this paper, interested readers can refer to (McCoy et al., 2020) for details.

However, as noted by (McCoy et al., 2020), it is the hierarchical rule that generalizes correctly to English, rather than the linear rule. As such, because the hierarchical rule relies on interpreting the tree-like structure of English syntax, it is necessary for high-performing models to correctly learn such tree-like syntactical structure.

[1]We make the code that we used, which is based on the work done by McCoy et al. (2020) and Yang and Wang (2024), available here: https://github.com/tylerzchen/kan_transformer

2 Related Work

2.1 Tree Transformers

The tree transformer (Shiv and Quirk) builds upon the traditional transformer architecture, as introduced in Vaswani et al. (2023). In contrast to the traditional transformer, which uses positional encodings that reflect the sequential position of the tokens, the tree transformer leverages the tree-like syntactic structure of English to create positional encodings based on a word's position within a syntactic tree designed to capture the structure of English.

In the tree transformer (Shiv and Quirk) encoding scheme, a token's positional encoding is therefore a function of its position within the syntactic tree created when parsing the sentence. As such, these tree positional encodings are able to better capture the tree-like syntactic structure of English.

In our work, we benchmark against two variants of tree transformers that are slightly modified from the original encoding system used by Shiv and Quirk. In particular, the ordering of our positional encodings differs from theirs, and we also test against two sets of tree parses: one without any tree nodes denoting syntactically empty nodes ("without EMPTY"), and one with such nodes provided ("with EMPTY"). The tree parses used in our work are provided by McCoy et al. (2020).

3 Background

The transformer, as introduced by Vaswani et al. (2023), has become the de facto architecture for natural language processing due to its combined features of attention and MLPs. This paper focuses on replacing MLPs with KANs. That is, we hope to replace the standard architecture of linear layers and non-linear activations with networks based on the Kolmogorov-Arnold Representation Theorem.

3.1 Kolmogorov-Arnold Representation Theorem

The Kolmogorov-Arnold representation theorem states that any continuous function

$$f(x_1, x_2, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}$$

can be expressed as a finite sum of continuous functions of one variable, as follows:

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \varphi_q \left(\sum_{p=1}^{n} \psi_{p,q}(x_p) \right),$$

where:

- $\varphi_q : \mathbb{R} \to \mathbb{R}$ and $\psi_{p,q} : \mathbb{R} \to \mathbb{R}$ are continuous functions,
- $x_1, x_2, \ldots, x_n$ are the input variables,
- n is the dimensionality of the input space.

This representation is valid for any continuous multivariate function f.

This is important for the world of machine learning, as a neural network is simply a multivariate function. Instead of using an MLP (multi-layer perceptron) composed of linear transformations and non-linear activations, we can use something that resembles the layout in the theorem. Since this theoretically can approximate any continuous function, we hypothesized that it should work very well in a transformer. In particular, we use the Kolmogorov-Arnold Representation Theorem to decompose a multivariate function as the sum of functions of linear combinations of univariate functions. In doing so, our architecture becomes simplified, because interaction effects between the different variables become simplified. However, the theorem guarantees that simplifying these interactions effects does not cost the neural network any expressive power; in other words, we are able to have simpler interactions without making it impossible for the network to learn certain kinds of functions.

3.2 Kolmogorov-Arnold Network Architecture

In this paper, we closely follow the KAN architecture proposed by Yang and Wang (2024). In their work, they modify the traditional transformer architecture by replacing the feed-forward layer with a group-KAN layer. In particular, in the base function of their group-KAN layer, they make use of rational functions as the base function for the layer. In contrast, in our work, we make use of a polynomial base function and benchmark its performance against those of other existing architectures.

4 Approach

To test the Kolmogorov-Arnold transformer, we constructed one by editing PyTorch's relevant transformer classes. Everywhere where an MLP was implemented, we replaced it with a KAN network. We used a seq2seq benchmarking task which involved taking a declarative sentence and generating the related question. For example, "The dog is barking" should produce "Is the dog barking". We tested this on both a test set and a generalization set included in the code.

4.1 Evaluation

4.1.1 Datasets

We evaluate on the test set and the generalization set in McCoy et al. (2020). The test set contains examples syntactically similar to the train set, while the generalization set contains examples that are syntactically different; in particular, in both the train and test sets, both *move-first* and *move-main* suffice to produce the correct output, so the model can therefore learn either rule to achieve high accuracy. However, in the generalization set, only the hierarchical rule (and not the linear rule) will produce the correct answer. Because the train set does not contain any samples in which both rules will produce different answers, the generalization set serves as a test as to how well the model is able to generalize to unseen examples. For that reason, we also therefore expect that the test set performance should be higher than the generalization set performance.

4.1.2 Metrics

It is intuitive to measure the full accuracy of the models, which is defined as the proportion of examples that the model got completely correct. However, McCoy et al. (2020) point out that full accuracy can be quite a strict measure. In practice, through manual inspection of model outputs, McCoy et al. (2020) find that the models often make small errors but get most of the sentence correct. However, full accuracy does not distinguish between getting a sentence wrong by one word and getting a sentence completely wrong, even though that is a meaningful distinction, because the former case would indicate that the model has learned *some* relevant linguistic structure that allows it to make a mostly-correct prediction. As such, in line with McCoy et al. (2020), we also use first-word accuracy, which is defined as the proportion of sentences in which the model got the first word correct. By relaxing the standard for accuracy, we thus capture cases in which the model is partially correct; this amounts to "partial credit" in evaluation. We use this particular metric for giving partial credit because it is useful in distinguishing between the two main rules that McCoy et al. (2020) find that models are likely to learn (or approximate): *move-first* and *move-main*. Because a model learning (or approximating) *move-first* will not have a high first-word accuracy in the generalization set, but a model learning (or approximating) *move-main* will likely achieve a high first-word accuracy, this metric serves as a useful way to distinguish between the two rules and thus is useful in measuring the model's generalization ability (McCoy et al., 2020).

Architecture	Test set first	Test set full
LSTM	0.996	0.733
Transformer	0.999	0.998
Tree transformer (without EMPTY)	1.000	0.993
Tree transformer (with EMPTY)	1.000	0.999
KAN with rational base function	0.998	0.972
KAN with polynomial base function	n/a	n/a

Architecture	Gen set first	Gen set full
LSTM	0.026	0.000
Transformer	0.06	0.000
Tree transformer (without EMPTY)	1.000	0.156
Tree transformer (with EMPTY)	1.000	0.847
KAN with rational base function	0.0	0.019
KAN with polynomial base function	n/a	n/a

Table 1: Evaluation results on the test and gen sets. We evaluate both the first-word accuracy and the full accuracy of each architecture.

5 Results

We tested the data on a test set and a generalization set. The results are in Table 1.

6 Discussion

6.1 Rational Base Function

For the rational base function KAN, we found that the model worked very well for the test set. The difference between the results for this and a standard transformer is likely due to hyperparameter tuning. We saw, however, that this model failed at the generalization test; in fact, all of which it generated was non-sensical. For example, by manually inspecting its outputs, we found that, in one case, the model generated "does some walrus that doesn't wait doesn't the vultures around her zebra" instead of the correct question "doesn't some walrus that does wait comfort the vultures around her". Grammatically, this is non-sensical and reflects the fact that the model does not generalize well to unseen data.

Our reasoning for this is that KANs are known to overfit data. This results from a KAN being able to fit a relationship remarkably strongly. This is also why we saw such great performance on the test set. This tells us that having some leniency in the model is ideal for generative tasks, or really any task. It is for this reason that we find the MLP (standard) transformer to outperform what should theoretically be a stronger transformer. Therefore, although the KAN is able to learn the train data well, it does not do well on unseen examples.

6.2 Polynomial Base function

We achieved no results with our polynomial base function. We found that the polynomial would diverge so much that the loss would diverge. We could have added a normalization factor, but that would be no different than using a specific case of a rational base function. With this in mind, the polynomial base function can only perform as well as the rational base function if we normalize it. Thus, the discussion follows from the previous section in the case that we use a normalized polynomial.

6.3 Validity of Assumptions Made

We note that the KAN architecture that we have used does not inherently imbue the model with the tree-like knowledge of syntax that is necessary to fully generalize to English; as a result, we expect that it behave similarly to the LSTM and traditional transformer in that they do not explicitly take in the tree-like structure of language. Because of this baked-in assumption about the sequence of language and the lack of accounting for the tree-like structure, we expect that the model perform moderately well on the test set but fail on the generalization set; this would imply that it learned something akin to the linear rule, rather than the hierarchical rule.

Implicitly, we also assume that our polynomial base function will be able to model language well. Because Taylor polynomials are widely known to be able to approximate analytic functions to arbitrary degrees of precision, we know that our KAN will, in theory, be able to learn any linguistic structure, especially because the Kolmogorov-Arnold theorem guarantees that the polynomial base function will be able to endow our KAN with the multivariate expressiveness that it needs.

7 Conclusion

In this study, we evaluate and benchmark different models on the task of English question formation. Our findings are applicable for researchers studying natural language processing or linguistics who seek to better understand the mathematical structure of language.

7.1 Future Work

Building upon the ideas presented in this work and previous works, we propose a few promising areas of investigation that may yield fruitful results for researchers in the future.

7.1.1 KANs with Different Base Functions

This work and previous work explore the use of polynomial and rational base functions. However, one may speculate on whether other base functions may yield different results. Perhaps, this would be true if we can find a base function that does not overfit as easily. In the following, we present several ideas.

1. **Fourier base functions**. By using a linear combination of sinusoidal functions of different periods, one can construct a different kind of base function that can model long-range cyclic dependencies. Although language is rarely perfectly cyclic, it often contains long-range dependencies that may be better modeled by sinusoidal functions than polynomials or rational functions. Intuitively, this is because polynomials and rational functions have the potential to overfit to one particular location or region and do not generalize well outside of that region. However, sinusoidal functions tend to replicate similar behavior outside of the training region, making them more suitable for generalization. If this is true, then we expect that KANs with Fourier base functions are good for tasks like next-word prediction.

2. **Piecewise Linear base functions.** In a similar vein, one might consider the generalizability

of linear functions and wonder whether it can be applied to a base function. However, because a purely linear base function would deny the model its nonlinear modeling capabilities, one may experiment with linear combinations of piecewise linear combinations of functions such as the ReLU function.

7.2 Training with Added Noise

The main problem that we faced with our design is overfitting. An idea to combat this is to add noise to the training data. In the context of language models, this would mean adding some randomness to the word embeddings. (Jain et al., 2023). Adding noise forces the model to be robust, as the training data is slightly different every time. This would help the KAN from overfitting. The question that is left is whether this defeats the purpose of this architecture. We utilize the Kolmogorov-Arnold Representation Theorem and its promised accuracy as the justification. If we are trying to now make the model less accurate to the training data, then it might not be worth it to stray away from an MLP in the first place.

7.2.1 Different Downstream Tasks

In this paper, we evaluated the models on the task of English question formation. However, a more general study on language should evaluate language models on a variety of tasks, such as NER, PoS tagging, machine translation, document summarization, and QA, to name a few. In particular, models should also be evaluated to determine how well they can capture (ultra)long-distance dependencies between words.

7.2.2 Incorporate Tree Parsing Directly

Because of the tree-like structure of syntax, it may be beneficial to test the effect of tree positional encodings, similarly to Shiv and Quirk. In doing so, the model would be able to better capture the syntactic information of English more easily without needing to explicitly construct an internal model of syntactic information. In particular, one might consider experimenting with different kinds of tree parses, or with having the model attempt to first learn a tree structure for language, and then using the learned tree structure to parse the sentences.

However, if the model is to learn the tree structure directly, then we predict that it will take massive amounts of training data that go beyond the dataset that we used. One would also have to define an appropriate objective to have it learn the tree-like structure properly.

7.2.3 Regularization

We find that the rational function KAN performs poorly on the generalization set, which suggests that some form of regularization may be useful. In addition to the currently accepted techinques of applying dropout (and performing a hyperparameter search for the best dropout probability), L_1, L_2 regularization, using ensembled models, and decreasing parameter count, we also propose the use of data augmentation for regularization. In particular, we propose a technique that we term *random permutation*. In random permutation, we add a small fraction of samples to the train set where the expected output is simply a random permutation of all of the words in the input.

As a data augmentation regularization technique, this is theoretically well-founded because it is agnostic to the true structure of English question formation, aside from the very simple assumption that all words appearing in the declarative also appear in the question form. Although one may argue that this does not take into account any domain-specific knowledge of English question formation, we argue that this is actually a desirable property because we want the model to learn this knowledge, and although we do have this knowledge for this particular task *a priori*, we do not, in general, have domain-specific *a priori* knowledge that can be applied to general natural language processing tasks. Furthermore, the goal of the generalization set is to measure the model's ability to generalize to unseen data; therefore, adding in domain knowledge could potentially constitute a leakage of the generalization set into the train set, which would render the measured generalization results inaccurate.

7.2.4 Gradient Clipping

We encountered the exploding gradient problem when attempting to train the polynomial KAN. Therefore, attempting again with gradient clipping could potentially resolve this issue. Gradient clipping refers to enforcing that gradients during backpropagation fall within a certain allowable range. If, at any time, the gradients are not within that range, then they are brought to the edge of the range closest to the gradient's true value. The benefit of this is that clipped gradients will ensure that the model does not run into the same exploding gradient problem. However, it also means that gradient descent will not be completely accurate, so it may lead to slower training. The amount of gradient clipping is a tunable hyperparameter that may be searched over.

References

Neel Jain, Ping yeh Chiang, Yuxin Wen, John Kirchenbauer, Hong-Min Chu, Gowthami Somepalli, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2023. Neftune: Noisy embeddings improve instruction finetuning. *Preprint*, arXiv:2310.05914.

R. Thomas McCoy, Robert Frank, and Tal Linzen. 2020. Does syntax need to grow on trees? sources of hierarchical inductive bias in sequence-to-sequence networks. *Transactions of the Association for Computational Linguistics*, 8:125–140.

Vighnesh Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. *Preprint*, arXiv:1706.03762.

Xingyi Yang and Xinchao Wang. 2024. Kolmogorov-arnold transformer. *Preprint*, arXiv:2409.10594.