Why Kolmolgorov-Arnold Networks (KAN) Work So Well: A Qualitative Explanation

Hung T. Nguyen[1], Olga Kosheleva[2], and Vladik Kreinovich[2]

[1]Department of Mathematical Sciences, New Mexico State University,
Las Cruces, New Mexico 88003, USA, hunguyen@nmsu.edu, and
Faculty of Economics, Chiang Mai University, Thailand
[2]University of Texas at El Paso, El Paso, Texas 79968, USA
olgak@utep.edu, vladik@utep.edu

1. What is a neural network: a brief reminder

- A general neural network consist of *neurons*, devices that transform input signals $x_1, \ldots, x_n$ into a signal $y = s(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)$.
- Here w_i are appropriate numerical coefficients known as *weights*, and $s(z)$ is a non-linear function known as the *activation function*.
- When we want to process some values $v_1, \ldots, v_m$:
  - some neurons take some (or all) of these values as inputs; these neurons form the first layer of the neural network;
  - other neurons process the output signals of these neurons; these neurons form the second layer, etc.
- At the end, the output signals of some neurons are returned as the results of data processing.
- These neurons form the last layer of the neural network.
- Neurons in each layer can also take inputs not only from the previous layer, but also from the pre-previous layer, etc.

2. What is a neural network (cont-d)

- Usually, in the neural networks:
  - the function $s(z)$ is fixed, and
  - the weights are adjusted so that in situations for which we know the desired output, the final output of the neural network is close to the desired one.

3. Where does the idea of a neural network come from?

- The main idea of a neural network came from human reasoning and data processing.
- This is performed by a network of interconnecting brain cells called *neurons*.
- In the first approximation, biological neurons can be described as performing the transformation

$$x_1, \ldots, x_n \to s(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n).$$

- This idea was solidified by the following *universal approximation* result – that:
  - any continuous function on a bounded domain
  - can be approximated, with any given accuracy, by a neural network.
- This universal approximation results holds for any non-linear activation function $s(z)$.

4. What Kolmorogov and Arnold proved

- In the 1950s, the famous Russian mathematician Andrei Kolmogorov and his then student Vladimir Arnold proved:
  - that neural networks not only can *approximate* continuous functions,
  - they can represent these functions *exactly*.
- Specifically, they proved that every continuous function $f(v_1, \ldots, v_m)$ on a bounded domain can be represented in the following form:

$$f(v_1, \ldots, v_m) = \sum_j f_j \left(\sum_{i=1}^{m} f_{i,j}(v_i) \right).$$

- Here, for each f, we need to find some continuous functions $f_i(z)$ and $f_{i,j}(z)$.

5. What Kolmorogov and Arnold proved (cont-d)

- This result solved one the 23 Hilbert's problems:
  - problems that in 1900, David Hilbert, the most renowned mathematician of his time,
  - formulated as the most important challenges for the 20th century mathematics.
- In terms of neural networks, this result means that every continuous function can be computed by the following multi-layer neural network:
  - first, neurons from the first layer input the original values v_i and return the values $z_{i,j} = f_{i,j}(v_i)$;
  - then, neurons from the second layer input the signals $z_{i,j}$ and return the signals $z_j = f_j(z_{1,j} + \ldots + z_{m,j})$;
  - finally, a linear neuron from the last layer uses the signals z_j to compute the final result $z_1 + z_2 + \ldots$

6. What Kolmorogov and Arnold proved (cont-d)

- The main difference between this universal representation result and the usual universal approximation result is that:
  - the approximation result considers neural networks in which all (or at least almost all) neurons have the same activation function,
  - while the universal representation result is only valid when we allow different activation functions for different neurons,
  - and different activations functions to represent different dependencies $f(v_1, \ldots, v_m)$.
- It can be proven that we cannot have the universal representation result if we consider only one fixed activation function.
- Indeed:
  - the set of all such functions can be described by finitely many parameters, i.e., has finite dimension, while
  - the set of all possible continuous functions is infinite-dimensional.

7. Kolmorogov-Arnold networks (KAN)

- The relation between the Kolmogorov-Arnold result and neurak networks was noticed in many papers.
- Recently, this relation became practical.
- Specifically, the Kolmogorov-Arnold result motivated computer scientists to consider neural networks in which:
  - instead of a fixed activation function $s(z)$,
  - we use a finite-parametric family of activation functions
    $$s(z, c_1, \ldots, c_\ell);$$
  - here, the values c_j can differ from one neuron to another.
- For example, we can use splines from a certain family.

8. Kolmorogov-Arnold networks (cont-d)

- During training:
  - not only the corresponding weights are adjusted,
  - but also the parameters $c_1, \ldots, c_\ell$ corresponding to different neurons.
- Because of the motivation, these networks were called *Kolmogorov-Arnold networks* (KAN, for short).

9. Kolmogorov-Arnold networks turned out to be very efficient

- The main objective of machine learning is to provide a good approximation to the desired dependency.
- In general, the more parameters we have, the more accurately we can approximate a given function; so:
  - a natural way to compare the quality of different approximation schemes is
  - to compare the approximation accuracy that these schemes provide when they use the same number of parameters.

  In these terms, Kolmogorov-Arnold networks turned out to be very efficient.
- For the same number of parameters:
  - they provide much better accuracy than the usual neural networks.
  - i.e., than neural networks that use a fixed activation function.

10. But why?

- A natural question is: why are the Kolmogorov-Arnold networks more efficient?
- In some cases, the dependence that we want to approximate is itself a composition of functions of the type $s_i(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)$ for different s_i.
- For example, a function $f(v_1, v_2) = \exp(\sin(\pi \cdot v_1) + v_2)$ can be computed as follows:
  - first, we compute $v = \sin(\pi \cdot x)$, which corresponds to $w_0 = w_2 = 0$, $w_1 = \pi$, and $s_1(z) = \sin(z)$; and
  - then, we compute the desired value as $\exp(v + v_2)$, with $w_0 = 0$, $w_1 = w_2 = 1$, and $s_2(z) = \exp(z)$.
- In this case, it is enough to use as many trained neurons as there are corresponding functions $s_i(z)$.
- However, Kolmogorov-Arnold neurons are successful in many other tasks as well.

11. But why (cont-d)

- How can we explain this empirical fact?
- Some preliminary attempts to explain this efficiency were presented.
- However, these attempts rely on an unspecified way of:
  - how the constant in the corresponding asymptotic expression
  - depends on the problem's dimension.
- Thus, these attempts do not yet form an explanation.
- In this talk, we use different ideas to explain why Kolmorogov-Arnold networks are, in practice, more efficient.
- At this stage, our explanation is qualitative.
- We hope that similar ideas can lead to a quantitative explanations of this difference.

12. Towards our explanation

- We want to explain why Kolmorogov-Arnold networks are efficient.
- For this purpose, let us first recall:
  - why neural networks are efficient in the first place, and
  - why deep networks – i.e., networks that consist of several layers – are more efficient than "shallow" (few-layer) neural networks.

13. Why neural networks: a brief reminder

- Many important computations require a large amount of computational steps – and thus, take a lot of time.
- A natural way to speed up computations is to perform some computations in parallel; similarly:
  - a natural way to speed up some human task such as cleaning the offices
  - is to have several people cleaning offices at the same time ("in parallel").
- First, some computational tasks are performed in parallel, the second group of tasks are performed in parallel, etc.
- In this scheme, the overall time needed for computation can be obtained by adding the computation times of different tasks.
- Thus, to make the resulting computation time as short as possible, it is desirable to make the tasks as fast as possible.

14. Why neural networks (cont-d)

- On each task, a computer performs some data processing.
- In a usual deterministic computer, the output signal of each computational step is uniquely determined by its inputs.
- In mathematical terms, this means that the output is a function of the inputs.
- So, selecting the fastest tasks means selecting the fastest-to-compute functions.
- In general, functions can be linear and non-linear.
- Linear functions are the easiest to compute, so it makes sense to consider linear functions.

15. Why neural networks (cont-d)

- However, we cannot only consider linear functions; indeed:
  - if on each stage of data processing, we compute linear functions, i.e., perform a linear transformation of inputs,
  - then the overall data processing will be the composition of linear transformations – which is known to be also linear
  - while in real life, we often have nonlinear dependencies.
- So, in addition to linear steps, we also need nonlinear steps.
- Which nonlinear functions are the easiest to compute?
- In general, the more inputs a function has, the more time it takes to compute this function.
- Thus, the fastest-to-compute are functions that have the smallest possible number of inputs – i.e., functions of one variable.

16. Why neural networks (cont-d)

- So, we end up with a layered computational scheme in which:
  - some layers compute linear combinations of inputs, while
  - other layers apply a nonlinear function to each input.
- We have already mentioned that a composition of linear transformations is still a linear transformation.
- Thus, it makes no sense to place linear layers one after another.
- Such two layers can be replaced by a single linear layer.
- Similarly, a composition $s_1(s_2(z))$ of two functions of one variable is still a function of a single variable.
- So, it makes no sense to place nonlinear layers one after another.
- Such two layers can be replaced by a single nonlinear layer.

17. Why neural networks (cont-d)

- Thus, in the resulting computation scheme:
  - each linear layer must be followed by a nonlinear layer, and
  - each nonlinear layer must be followed by a linear layer.
- We can thus divide the whole computation scheme into pairs of layers consisting of:
  - a linear layer – that computes a linear combination $z = w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n$ of the inputs
  - followed by a nonlinear layer – that compute the value $y = s(z)$.
- Substituting the expression for z into this formula, we conclude that each pair of layers computes the expression $y = s(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)$.
- This is exactly what is usually by a neuron.
- So, we indeed explained why we should use neural networks.

18. Why deep neural networks: a brief reminder

- Whatever approximation scheme we use, in a computer, parameters are represented as a sequence of bits, i.e., of 0s and 1s.
- Let N denoted the overall number of bits that we use to store all these parameters.
- Then, we can have 2^N possible combinations of these parameters.
- So, in principle, we can represent 2^N different approximating functions.
- The more approximating functions we have, the more accurately we can approximate any function.
- For example, suppose that we are approximating numbers from the interval $[0, 1]$.
- If we use only one approximating value, then the best we can do is to use the value 0.5.
- This enables us to approximate any number with accuracy 0.5.

19. Why deep neural networks (cont-d)

- If we are allowed to use two approximating values, then we can select the values 0.25 and 0.75.
- This will enable us to approximate any number with accuracy 0.25, etc.
- In principle:
  - if we have P real-valued parameters and each of them consists of B bits,
  - then to store all these parameters, we need to use $P \cdot B$ bits.
- However, this does not necessarily mean that we can have $2^{P \cdot B}$ different functions.
- For example, suppose that to process m inputs, we use a shallow neural network,
- So, we have K neurons followed by a linear combination of their output signals.

20. Why deep neural networks (cont-d)

- Then:
  - describing each neuron requires $m + 1$ parameters $w_0, \ldots, w_m$, to the total of $K \cdot (m + 1)$, and
  - describing the linear neuron is the last layer requires $K + 1$ parameters $W_0, W_1, \ldots, W_K$.
- So overall we need $P = K \cdot (m + 1) + K + 1$ parameters.
- This means that we have $N = P \cdot B$ bits, so we have 2^N possible combinations of these bits.

21. Why deep neural networks (cont-d)

- However, some of these combinations lead to the same function.
- Indeed, if we permute the neurons, the resulting data processing will lead to the same result, but the bit sequences will be different.
- These are $K! = 1 \cdot 2 \cdot \ldots \cdot K$ different permutations of K neurons.
- So, for each bit sequence, $K!$ different bits sequences lead to the same approximating function.
- Thus, the number of different approximating functions produced by a shallow network can be obtained by dividing 2^N by $K!$.
- For large K, the factorial $K!$ is huge, so the ratio $2^N/K!$ is much smaller than the ideal value 2^N.
- A natural way to limit this decrease is to place the neurons in several layers.

22. Why deep neural networks (cont-d)

- If we have L layers with K/L neurons in each layer, then we can still perform $\left(\frac{K}{L}\right)!$ permutations in each layer.
- The resulting number of possible combinations of L permutations equal to $\left(\left(\frac{K}{L}\right)!\right)^L$.
- Let us show that this indeed reduces the number of bit combinations leading to the same function – and thus, increases the number of approximating functions.
- Indeed, it is known that factorial is well approximated by a formula

$$n! \approx \left(\frac{n}{e}\right)^n.$$

23. Why deep neural networks (cont-d)

- In this approximation, for the shallow neural network, we get

$$K! \approx \left(\frac{K}{e}\right)^K, \text{ while } \left(\frac{K}{L}\right)! \approx \left(\frac{K}{L \cdot e}\right)^{K/L}; \text{ thus}$$

$$\left(\left(\frac{K}{L}\right)!\right)^L \approx \left(\left(\frac{K}{L \cdot e}\right)^{K/L}\right)^L = \left(\frac{K}{L \cdot e}\right)^K = \frac{1}{L^K} \cdot \left(\frac{K}{e}\right)^K.$$

- By comparing these two values, we see that

$$\left(\left(\frac{K}{L}\right)!\right)^L \approx \frac{1}{L^K} \cdot K!.$$

- So, for deep L-layered neural networks:
  - the number of duplicate bit combinations is indeed much smaller
  - than for the shallow neural network with the same number of parameters.

24. Why deep neural networks (cont-d)

- Thus, with the same number of parameters, by using a deep network, we can represent much more functions.
- So, we can approximate any given function much more accurately.

25. So why Kolmogorov-Arnold neural networks?

- Now, we are ready to analyze the difference between:
  - the usual deep neural networks – in which the activation function for each neuron is fixed from the very beginning – and
  - a Kolmogorov-Arnold neural network in which for each neuron, the activation function is adjusted based on the training results.
- In the latter case, to described each neuron:
  - in addition to specifying $n+1$ weights w_i,
  - we also need to specify ℓ values of the parameters c_i.
- So, to specify each neuron, we need to use $\alpha \stackrel{\text{def}}{=} \dfrac{n+1+\ell}{n+1}$ more parameters than in the usual deep learning.
- If we keep the same number of neurons in each layer, then the overall number of parameters is proportional to the number of layers.
- Now each neuron requires α times more parameters to describe.

26. So why Kolmogorov-Arnold neural networks (cont-d)

- So, for the same overall number of parameters, we can use α times fewer layers $L' \stackrel{\text{def}}{=} \dfrac{K}{\alpha}$; thus:
  - instead of the original amount $D = \left(\left(\dfrac{K}{L}\right)!\right)^{L}$ of duplicate bit strings,
  - we now have $D' = \left(\left(\dfrac{K}{L}\right)!\right)^{L/\alpha}$ duplicate strings.
- One can easily see that $D' = D^{1/\alpha}$ for $\alpha > 1$.
- For example, for $\alpha = 2$, we get $D' = \sqrt{D}$.
- This is much smaller than the original number of duplicate binary strings.

27. So why Kolmogorov-Arnold neural networks (cont-d)

- Thus, with the same number of parameters:
  - by using a Kolmorogov-Arnold network,
  - we can represent much more functions and thus,
  - approximate any given function much more accurately.
- So, we have indeed explained the effectiveness of such networks.

28. Acknowledgments

This work was supported in part by:

- National Science Foundation grants 1623190, HRD-1834620, HRD-2034030, and EAR-2225395;
- AT&T Fellowship in Information Technology;
- program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and
- a grant from the Hungarian National Research, Development and Innovation Office (NRDI).