Kolmogorov-Arnold Networks

jhl

@c/sim-aaa · @ucsd

10 June 2024

Welcome to the inaugural Sim-Eval literature review!

I'm trying to keep this informal – teaching is the best way of learning, and we all benefit from sharing knowledge. (I also only had a weekend to prepare.) **Some bookkeeping notes:**

- Expect a bit more of the underlying intuition here, instead of the mechanical $\varepsilon - \delta|_{n\to\infty}$ aspects of the proofs.
- For simplicity, assume that the functions we care about are *of multiple variables* and *scalar-valued*, i.e.

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}$$

- This talk is motivated by a recent paper by Liu, Wang et al. It is accessible at `https://arxiv.org/pdf/2404.19756v1`.

A gentle (re)introduction to the multi-layer perceptron

The multi-layer perceptron (MLP) is the fundamental building block of so-called deep neural networks. Deep refers to the large "depth" of the network, i.e. the number of layers.

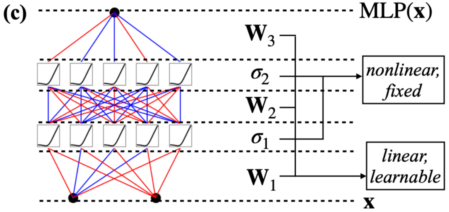


Figure: *A deep multi-layer perceptron network.*

Canonically, a "shallow" multi-layer perceptron is *two layers*. (This will be important for the literature!) A multi-layer perceptron in the shallow case is represented as

$$\hat{f} \colon \mathbb{R}^n \longrightarrow \mathbb{R}$$

$$\hat{f}(x) = \sum_{i}^{N(\varepsilon)} a_i \sigma \left[\mathbf{W}_i \mathbf{x} + b_i \right]$$

Note the width $N(\varepsilon)$ is a function of the precision ε.

$$\hat{f}(x) = \sum_{i}^{N(\varepsilon)} a_i \sigma \left[\mathbf{W}_i \mathbf{x} + b_i \right]$$

where:

b_i	is the bias *(affine!)*
$\mathbf{x}$	is the i th input vector
$\mathbf{W}_i$	is the i th weight matrix *(learned!)*
σ	is the activation function *(e.g. ReLU, sigmoid,* tanh*...)*
a_i	are elements of the outermost weight matrix
$N(\varepsilon)$	is the number of neurons

In the shallow case, a_i can be denoted by a row vector.

$$\hat{f}(x) = \sum_{i}^{N(\varepsilon)} a_i \sigma \left[\mathbf{W}_i \mathbf{x} + b_i \right]$$

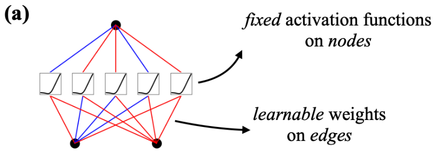

(a)
fixed activation functions on *nodes*
learnable weights on *edges*

Figure: *A shallow 2-layer multi-layer perceptron.* $d = 2$, $n = 5$.

More generally, for deep ($d > 2$) networks, a multi-layer perceptron is really just a composition of (affine!) linear transformations separated by non-linear activations.

$$MLP(\mathbf{x}) = [\mathbf{W}_d \circ \sigma_d \circ \mathbf{W}_{d-1} \circ \sigma_{d-1} \circ \dots \mathbf{W}_1 \circ \sigma_1] (\mathbf{x})$$

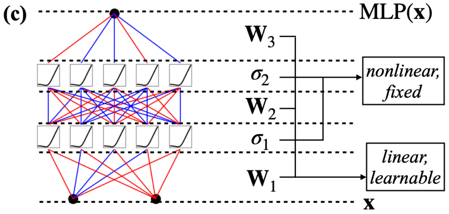
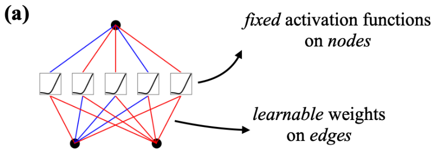


Figure: *A deep multi-layer perceptron network.*

$$\hat{f}(x) = \sum_{i}^{N(\varepsilon)} a_i \sigma \left[\mathbf{W}_i \mathbf{x} + b_i \right]$$

Question: What can a multi-layer perceptron represent?



(a)
fixed activation functions on *nodes*
learnable weights on *edges*

Figure: *A shallow 2-layer multi-layer perceptron.* $d = 2$, $n = 5$.

Universal Approximation

Question: What can a multi-layer perceptron represent?
Answer Anything!*

Let $D \subset \mathbb{R}^n$ be compact[1]. $f \colon D \longrightarrow \mathbb{R}$ be an *arbitrary nonlinear function*, and let $\hat{f} \colon D \longrightarrow \mathbb{R}$ denote a shallow (*n.b.* 2-layer) multi-layer perceptron, denoted by

$$\hat{f}(x) = \sum_{i}^{N(\varepsilon)} a_i \sigma \left[\mathbf{W}_i \mathbf{x} + b_i \right]$$

where $N(\varepsilon)$ is the *number of neurons*. In the shallow case this is = the width.

[1] Compact denotes some notion of "closed and bounded" – the n-dimensional equivalent of a closed interval $[a, b] \subset \mathbb{R}$.

Let $D \subset \mathbb{R}^n$ be compact. $f\colon D \longrightarrow \mathbb{R}$ be an *arbitrary nonlinear function*, and let $\hat{f}\colon D \longrightarrow \mathbb{R}$ denote a shallow (*n.b.* 2-layer) multi-layer perceptron, denoted by

$$\hat{f}(x) = \sum_{i}^{N(\varepsilon)} a_i \sigma \left[\mathbf{W}_i \mathbf{x} + b_i\right]$$

Theorem

Universal approximation (2-layer network). *For arbitrary* $\varepsilon \in \mathbb{R} > 0$, *there exists* $N(\varepsilon)$ *such that*

$$|f(x) - f(\hat{x})| \leq \varepsilon$$

$$\hat{f}(x) = \sum_{i}^{N(\varepsilon)} a_i \sigma\left[\mathbf{W}_i \mathbf{x} + b_i\right]$$

Theorem

Universal approximation (2-layer network). *For arbitrary $\varepsilon \in \mathbb{R} > 0$, there exists $N(\varepsilon)$ such that*

$$|f(x) - \hat{f(x)}| \leq \varepsilon$$

So we can model basically anything! But...

Theorem

Universal approximation (2-layer network). *For arbitrary* $\varepsilon \in \mathbb{R} > 0$, *there exists* $N(\varepsilon)$ *such that*

$$|f(x) - \hat{f(x)}| \leq \varepsilon$$

...the practical question is:

What is $N(\varepsilon)$?

Is it $O(\log \varepsilon^{-1})$? $O(\varepsilon^{-1})$? $O(\exp \varepsilon^{-1})$?

Neural scaling

What is $N(\varepsilon)$?
We don't know, in general! The universal approximation theorem guarantees no bounds on N. For deep networks, we do know that it's possibly poorly behaved ($N \propto \exp(d)$, the layer depth of the network).

Why does this make sense? Because we are fitting a "mostly linear" model to an "arbitrary non-linear" function. We need "a lot of linear pieces" to get good at modeling funky nonlinear functions.

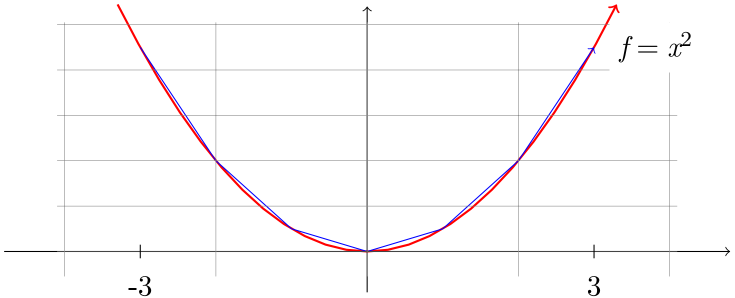


Figure: *It takes a lot of line segments to approximate this quadratic, and as soon as we leave $[-3, 3]$, the error in our approximation blows up!*

Enter the notion of *neural scaling laws*.

Neural scaling laws formalize the notion of "mostly linear things approximate nonlinearities inefficiently" – in particular, we can generally say that the *training*[2] loss ℓ decreases according a power-law regime:

$$\ell \propto N^{-\alpha}$$

where α is the *scaling exponent* and N is the number of parameters. In essence, to decrease the loss linearly requires an exponential increase in the number of parameters.

[2] i.e. overfits

In essence, to decrease the loss linearly requires an exponential increase in the number of parameters.
Fundamentally, if the underlying process is nonlinear, then we are basically trying to fit a big piecewise linear function, and then combining them with a fixed nonlinearity (ReLU, sigmoid, tanh, &c).

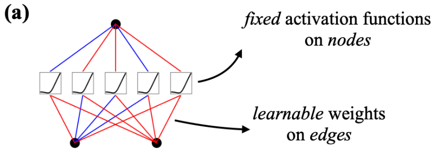
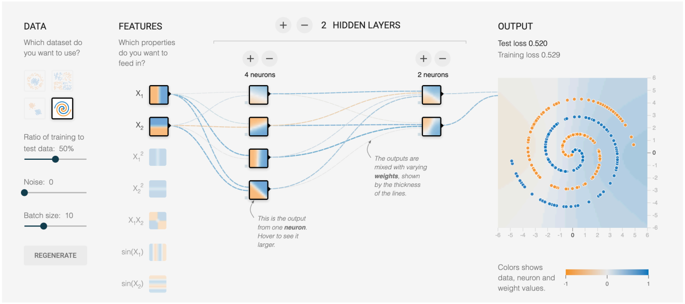

(a)
fixed activation functions on *nodes*
learnable weights on *edges*

Figure: *The shallow perceptron again.* The nodes apply a fixed nonlinear activation to each input, which is a learned affine linear function over the incoming edges.

A practical example: `go/tensorflowplayground`



DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
+ − 2 HIDDEN LAYERS
+ −
4 neurons
+ −
2 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.520
Training loss 0.529
Colors shows data, neuron and weight values.
-1
0
1

Figure: *Separating linearly inseparable data, with linear functions?*

Idea: Why not *learn the nonlinearity*?

Using linear piecewise bits to represent a nonlinear function seems impractical. Besides, we already use nonlinear basis functions to fit models (e.g. polynomial degree-n regression, exponential regression, spline fitting).

This is the basic idea behind the Kolmogorov-Arnold network.

Kolmogorov-Arnold Networks

The fundamental architectural difference (other than learning the nonlinear 'activation') between the multi-layer perceptron and the Kolmogorov-Arnold network is that nodes and edges are flipped:

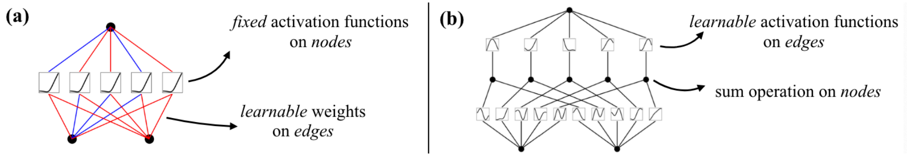

(a)
fixed activation functions on *nodes*
learnable weights on *edges*
(b)
learnable activation functions on *edges*
sum operation on *nodes*

Figure: *MLP (right) vs. KAN (left) in the shallow case.*

Both of these are two-layer architectures.

Kolmogorov-Arnold Representation

The multi-layer perceptron was underpinned by the Universal Approximation theorem. The Kolmogorov-Arnold network is similarly underpinned by the Kolmogorov-Arnold representation theorem (surprise). Let f be a continuous function defined on a closed hypercube, i.e. $f\colon [0,1]^n \subset \mathbb{R}^n \longrightarrow \mathbb{R}$.

Theorem

Kolmogorov-Arnold representation. *f admits a representation that is the sum of continuous functions of a single variable. In particular: there exist $\phi_{q,p} : [0,1] \longrightarrow \mathbb{R}$ and $\Phi_q : \mathbb{R} \longrightarrow \mathbb{R}$ where*

$$f(x) = f(x_1, x_2, \dots x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$$

Theorem

Kolmogorov-Arnold representation. *f admits a representation that is the sum of continuous functions of a single variable. In particular: there exist $\phi_{q,p} : [0,1] \longrightarrow \mathbb{R}$ and $\Phi_q : \mathbb{R} \longrightarrow \mathbb{R}$ where*

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

Well that's abstract and unhelpful! What does it mean?

Rather surprisingly, it tells us that *every multivariable function defined on a compact set can be written with a collection of univariate functions, composed and added together*! This is a pretty remarkable result:

$$x \in [0,1]^n \implies$$

$$\underbrace{f(x) = f(x_1, x_2, \ldots x_n)}_{\text{multivariate}} = \sum_{q=0}^{2n} \underbrace{\Phi_q}_{\text{univariate}} \left(\sum_{p=1}^{n} \underbrace{\phi_{q,p}}_{\text{univariate}} (x_p) \right)$$

$x \in [0,1]^n \implies$

$$\underbrace{f(x) = f(x_1, x_2, \ldots x_n)}_{\text{multivariate}} = \sum_{q=0}^{2n} \underbrace{\Phi_q}_{\text{univariate}} \left(\sum_{p=1}^{n} \underbrace{\phi_{q,p}}_{\text{univariate}} (x_p) \right)$$

There is a catch: we have no guarantees on how nicely-behaved these univariate functions are. It could well be the case that our ϕ, Φ are nondifferentiable or even fractal.

Fortunately, in practice, (it turns out) if you're modeling a real-life process, the odds are good that they will be (reasonably) well-behaved.

In addition, (in much the same way as a deep MLP network,) we can stack layers of these KAN networks together.

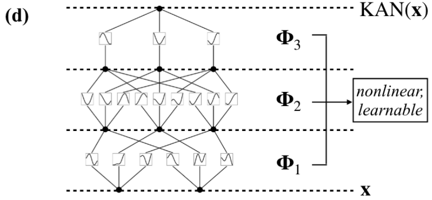


Figure: *A deep KAN.*

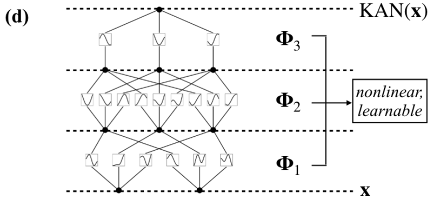


Figure: *A deep KAN.*

Empirically, it turns out these have high approximating power (among other benefits!). A deep KAN admits a very simple representation:

$$\mathrm{KAN}(x) = (\phi_n \circ \phi_{n-1} \circ \ldots \circ \phi_1)(x)$$

Compare to the deep MLP formulation:

$$MLP(\mathbf{x}) = \left[\mathbf{W}_d \circ \sigma_d \circ \mathbf{W}_{d-1} \circ \sigma_{d-1} \circ \ldots \mathbf{W}_1 \circ \sigma_1\right](\mathbf{x})$$

From the previous discussion about neural scaling, we might expect a KAN to be *more efficient* at representing nonlinear processes than a deep MLP.

Is this true?