Kolmogorov–Arnold Networks

Yifeng He

May 15, 2024

Section 1

Background and Motivation

Curve Fitting with MLP

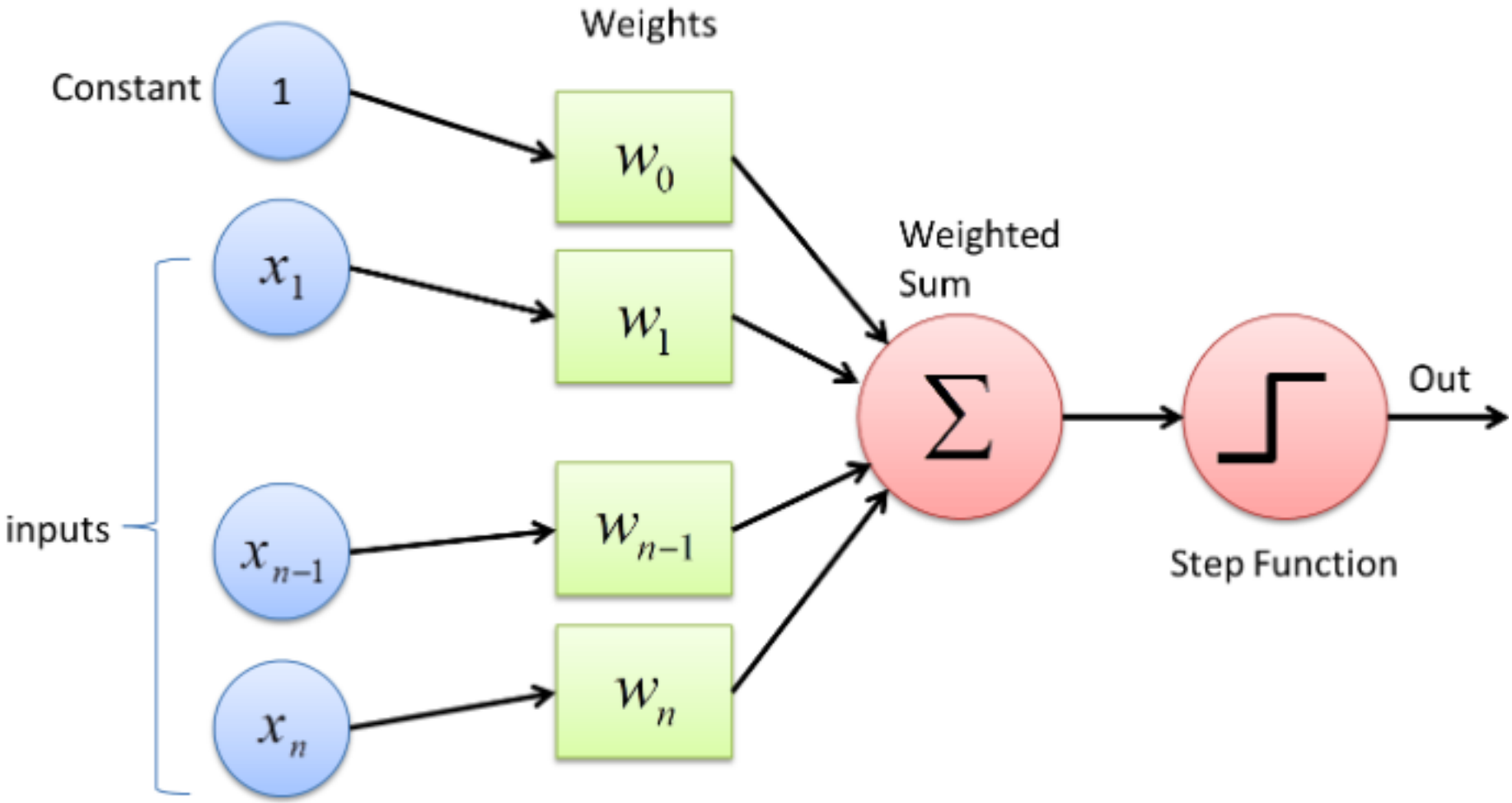


Figure 1: Multilayer Perceptron Network

Is there other ways to represent a curve?

Interpolation and the Lagrange Polynomial

We are given a set of points $A = \{(x_i, y_i) | x_i, y_i \in \mathbb{R}\}$, we want to find a (continuous) function $f : \mathbb{R} \to \mathbb{R}$ such that f goes through all points in A. Then f can be expressed in polynomial form

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

where n is a nonnegative integer and $a_0, \ldots, a_n$ are real constants. We say this is a nth degree approximation of f.

Weierstrass Approximation Theorem (WAT). Suppose f is defined and continuous on [a, b]. For each $\epsilon > 0$, there exists a polynomial P(x), with the property that

$$|f(x) - P(x)| < \epsilon, \forall x \in [a, b].$$

Why Polynomial Interpolation?

WAT $\Longrightarrow$ uniform approximation.

Given any function, defined and continuous on a closed and bounded interval, there exists a polynomial that is as "close" to the given function as desired.

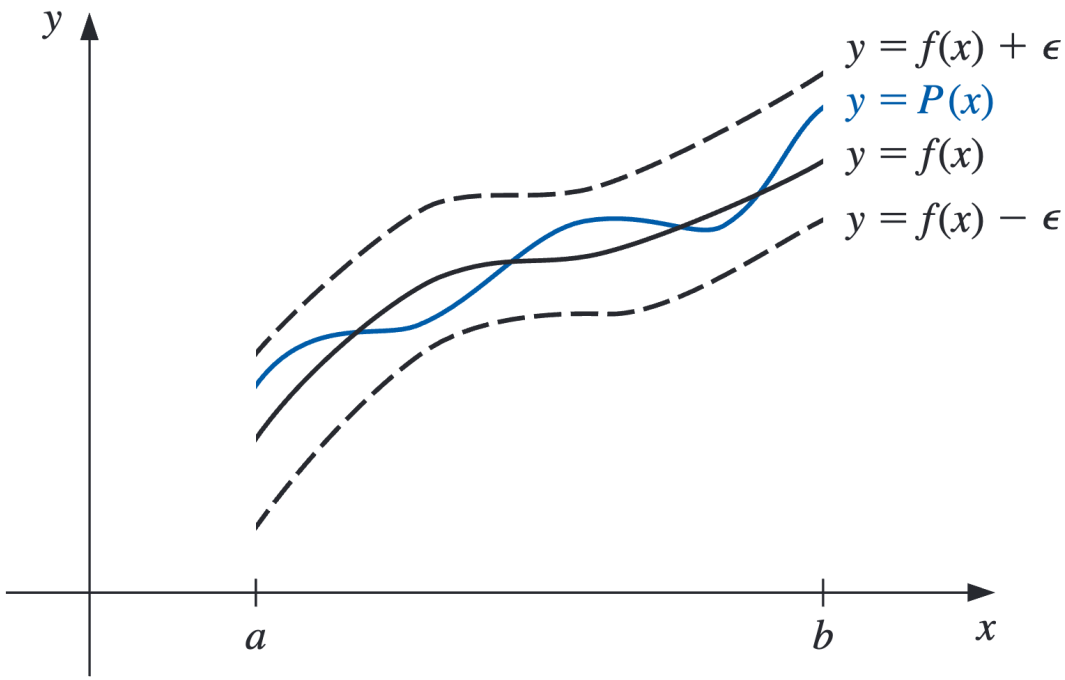


Figure 2: WAT

Why it is not widely used in ML?

To pass through n points

$$\{x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_{n-1}, f(x_{n-1})\}$$

we need a polynomial of degree $n-1$.

However, real datasets often contain *a lot* of points. When the degree n is very large, polynomials can oscillate erratically.

Cubic Spline Interpolation

One way to overcome the oscillate problem is to limit the degree n. That is, divide f into different grids and interpolate a polynomial with smaller degree for each section.

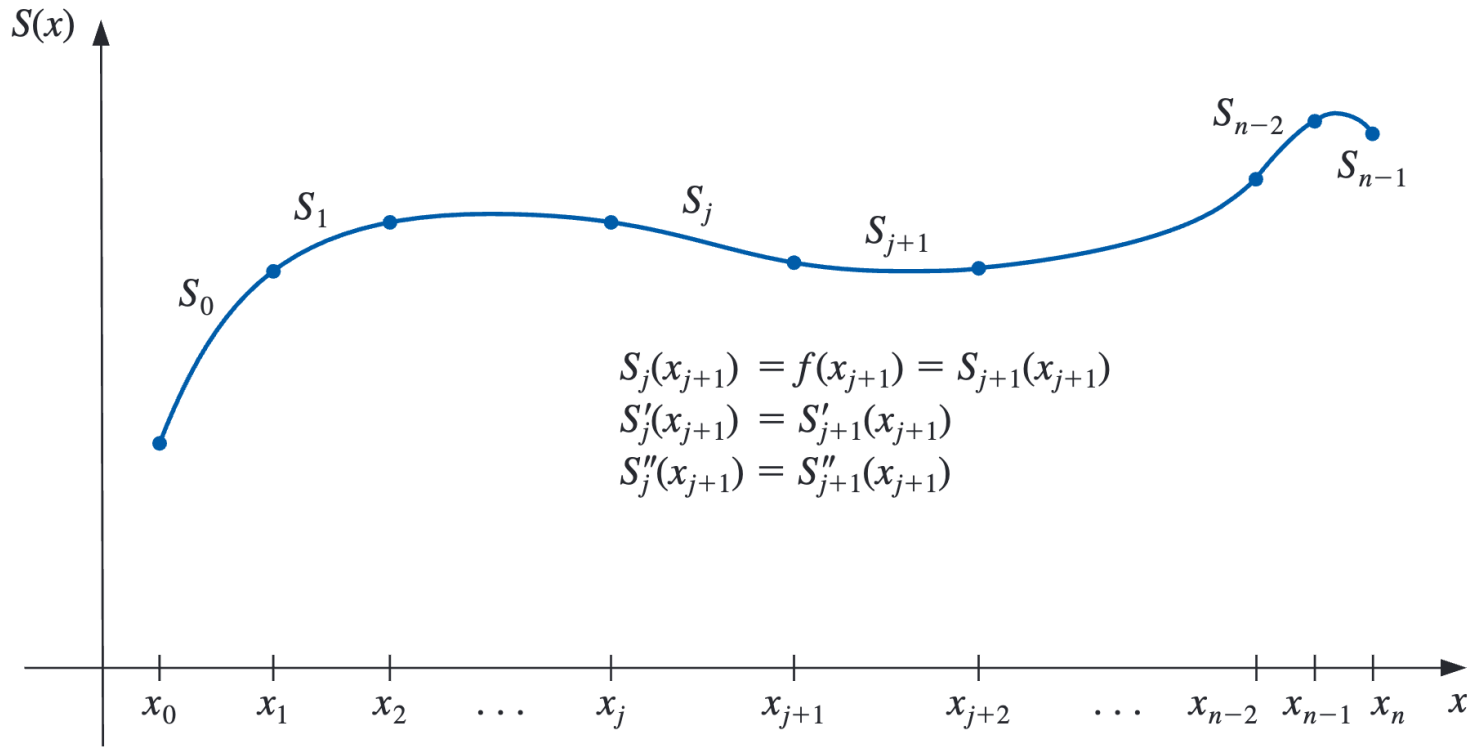


Figure 3: Spline in Sub-intervals

Remaining Problem

In real applications, data is **multivariate**. For example, in the formula of linear regression

$$\hat{y} = X\beta + \epsilon$$

or

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i$$

for $i = 1, \ldots, n$. i.e. You can (and should) use multiple predictors/features.

However, polynomial interpolation is **univariate**.

Section 2

KAN: Kolmogorov–Arnold Networks

Kolmogorov-Arnold Representation Theorem

If f is a multivariate continuous function on a bounded domain, then f can be written as a *finite composition* of continuous functions of a single variable and the binary operation of addition.

That is, for a smooth $f : [0,1]^n \to \mathbb{R}$,

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

where $\phi_{q,p} : [0,1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$.

KRT to Machine Learning

Recall WAT, $f : [a, b] \to \mathbb{R}$ is just a parametric form of $\phi_{q,p}$ with fixed $a = 0, b = 1$. Then we can represent a multivariate function with just spline polynomials!

But how can we let the machine learn the spline polynomials $\phi_{q,p}$?

Some Linear Algebra

Recall linear regression, where we learn a *linear transformation* of the predictors. To apply back propagation, we need some *parameterized* form of the splines.

Recall from linear algebra, let V be a vector space of finite dimension n and

$$B = \{b_1, \ldots, b_n\}$$

be the basis of V. Then every vector v in V can be written as

$$v = \lambda_1 b_1 + \cdots + \lambda_n b_n$$

where λ_is are *constants*.

B-spline

The set of all cubic splines on $[0, 1]$ form a vector space V. Let

$$B = \{B_1, \ldots, B_n\}$$

be the basis functions of V, then every spline function ϕ can be written as

$$\phi(x) = \sum_{i=0}^{n} \lambda_i B_i(x).$$

Then we have a parametrized form of non-linear functions.

In other words, now we can use back propagation to learn activation functions! :)

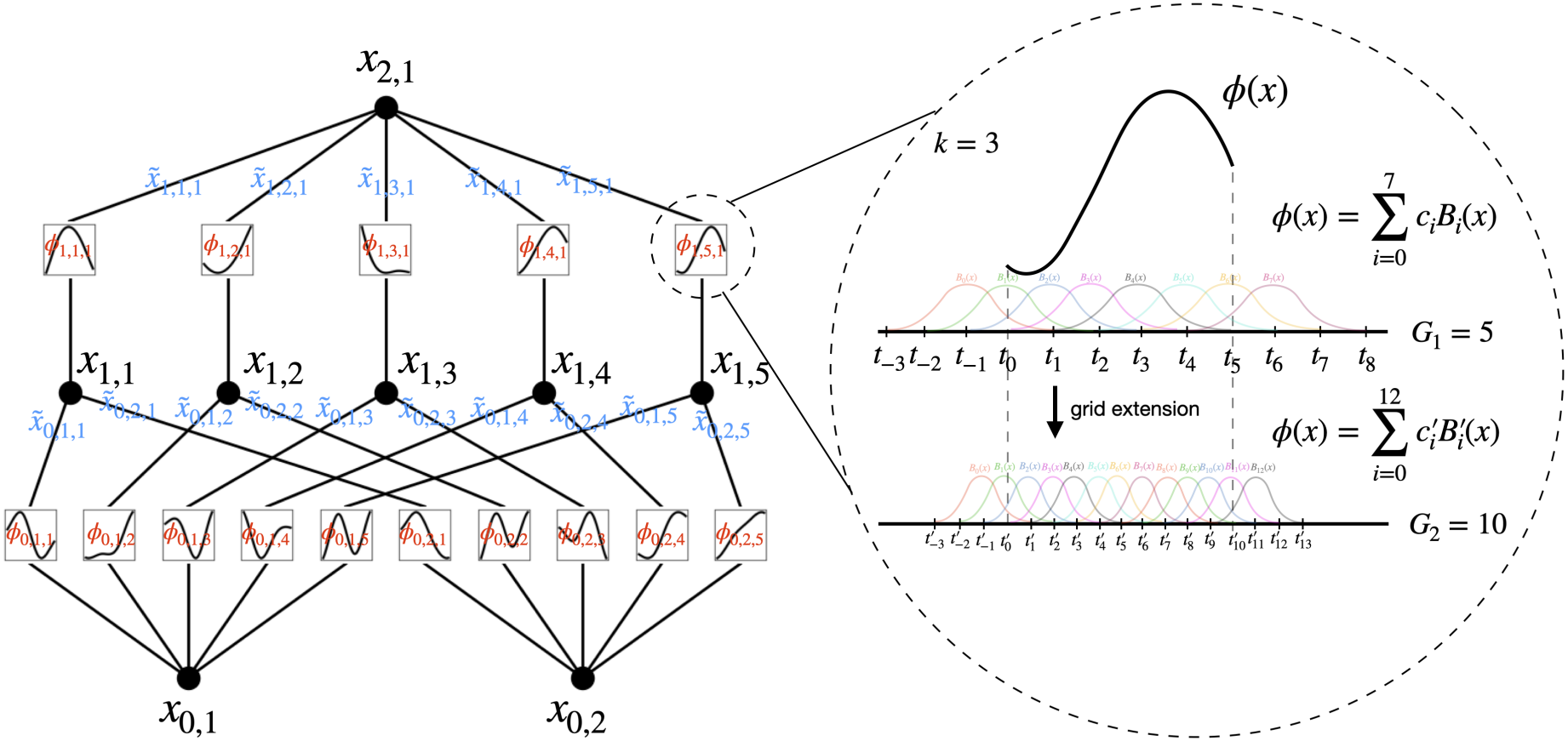


Figure 4: Left: Notations of activations that flow through the network. Right: an activation function is parameterized as a B-spline, which allows switching between coarse-grained and fine-grained grids.

Kolmogorov-Arnold Representation Network

Now it is clear what is a layer in KAN is $\Phi = \{\phi_{q,p}\}$ where $p = 1, 2, \cdots, n_{in}$ and $q = 1, 2, \cdots, n_{out}$, is just a set of *learned* non-linear activations.

Then KAT is just a 2 layer KAN with first layer $n_{in} = n$ and $n_{out} = 2n + 1$, and the second layer with $n_{in} = 2n + 1$ and $n_{out} = 1$. In this case, we say this KAN has shape $[n, 2n + 1, 1]$.

Then we can start stacking layers. :smiley:

$$KAN(X) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_1 \circ \Phi_0)X.$$

Compared to MLP

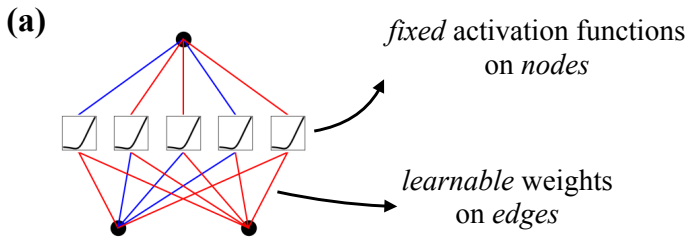
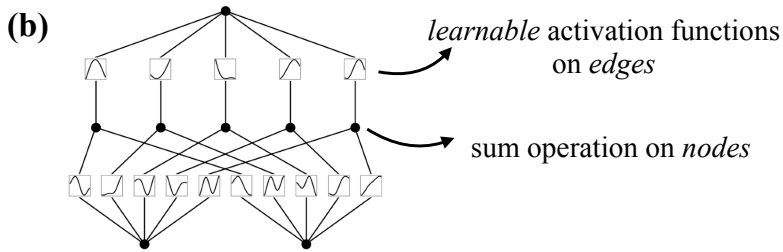
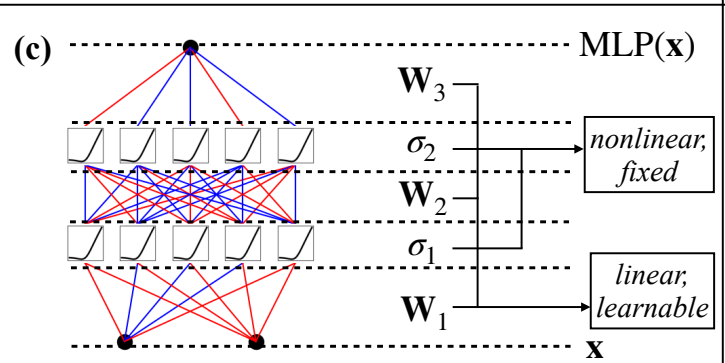
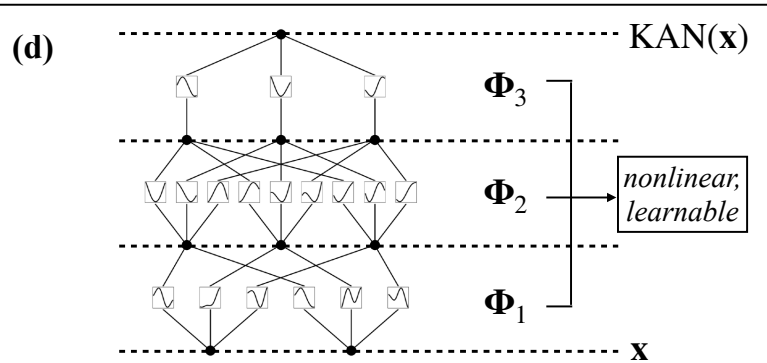
Model	**Multi-Layer Perceptron (MLP)**	**Kolmogorov-Arnold Network (KAN)**
Theorem	**Universal Approximation Theorem**	**Kolmogorov-Arnold Representation Theorem**
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)		
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)		

Figure 5: Multi-Layer Perceptron vs. Kolmogorov-Arnold Network

Section 3

Experiments

Making KAN more accurate by Grid Extension

From "neural scaling laws", we can improve MLP-based networks by making them larger, but we have to *retrain* the network. A spline can be made arbitrarily accurate to a target function as the grid can be made arbitrarily *fine-grained*.

- Does KAN inherited this notation of "fine-graining"?
- If so, can this be done during training?

Say we have a activation function f, coarse-grained grid G_1 and fine-grained G_2. Then we have

$$f_{coarse}(x) = \sum_{i=0}^{G_1+k-1} c_i B_i(x)$$
$$f_{fine}(x) = \sum_{j=0}^{G_2+k-1} c_j' B_j'(x)$$

Then we can calculate the weights c_j' *directly* from c_i by minimizing the distance between $f_{fine}(x)$ and $f_{coarse}(x)$.

$$\{c_j'\} = \underset{\{c_j'\}}{\operatorname{argmin}} \underset{x\sim p(x)}{\mathbb{E}} \left(\sum_{j=0}^{G_2+k-1} c_j' B_j'(x) - \sum_{i=0}^{G_1+k-1} c_i B_i(x) \right)^2.$$

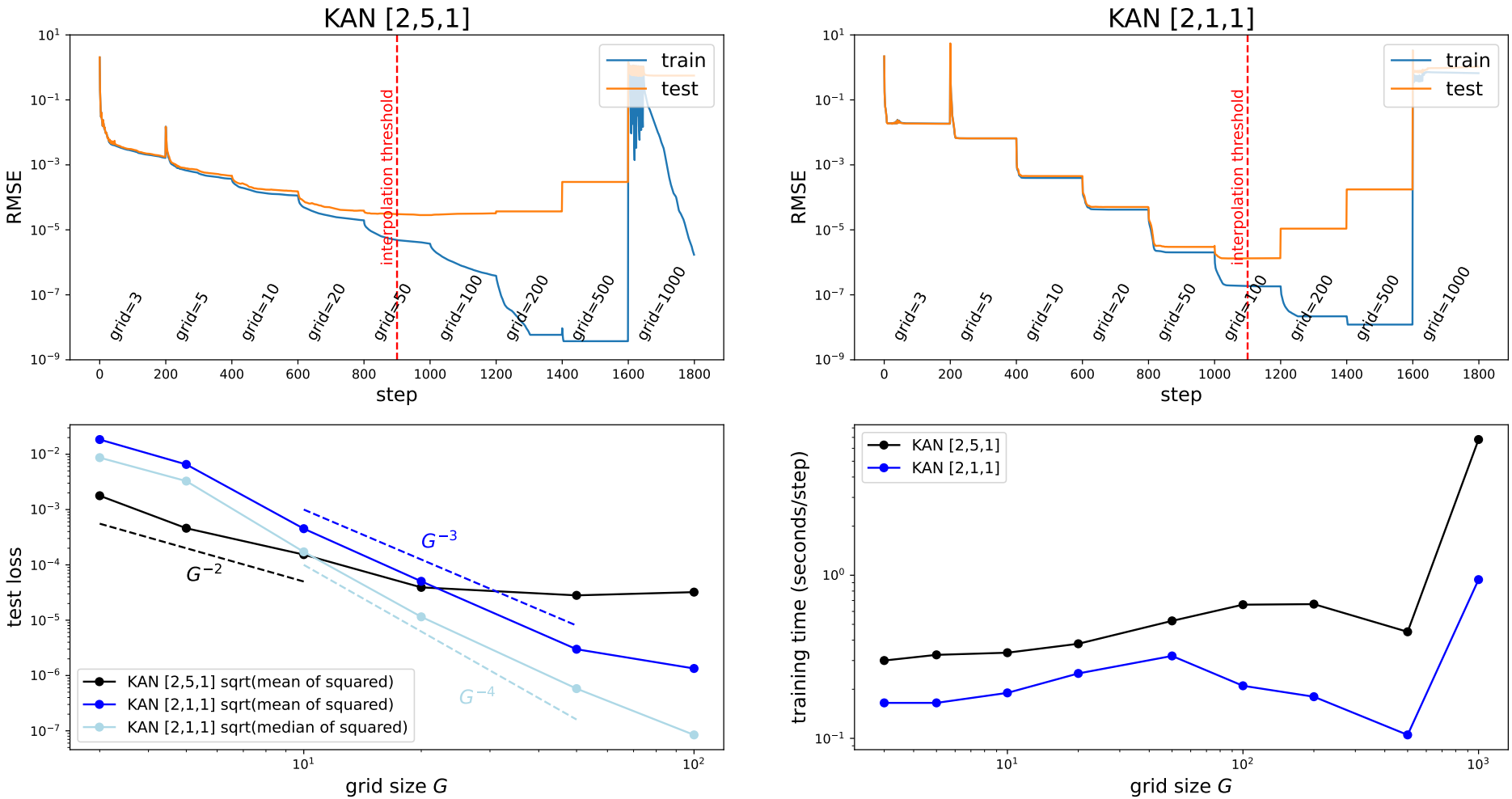


Figure 6: We can make KANs more accurate by grid extension during training.

Small KANs generalize better.

Overfitting? Can we regularize KANs?

Recall in linear regression, we can use LASSO (l_1 norm) to set some coefficients to zero for **sparsity**.

Problem: l_1 norm is not defined for activation functions ϕ, so let's define the norm fo a as its average magnitude over its N_p inputs, i.e.,

$$|\phi|_1 \equiv \frac{1}{N_p} \sum_{s=1}^{N_p} \left| \phi(x^{(s)}) \right| .$$

Then for a KAN layer Φ with n_{in} inputs and n_{out} outputs, it is just the sum

$$|\Phi|_1 \equiv \sum_{i=1}^{n_{in}} \sum_{j=1}^{n_{out}} |\phi_{i,j}|_1 .$$

In addition, we define the entropy of Φ to be

$$S(\Phi) \equiv -\sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} \frac{|\phi_{i,j}|_1}{|\Phi|_1} \log \left(\frac{|\phi_{i,j}|_1}{|\Phi|_1} \right).$$

The total training objective ℓ_{total} is the prediction loss ℓ_{pred} plus L1 and entropy regularization of all KAN layers:

$$\ell_{\text{total}} = \ell_{\text{pred}} + \lambda \left(\mu_1 \sum_{l=0}^{L-1} |\Phi_l|_1 + \mu_2 \sum_{l=0}^{L-1} S(\Phi_l) \right),$$

where μ_1, μ_2 are relative magnitudes usually set to $\mu_1 = \mu_2 = 1$, and λ controls overall regularization magnitude.

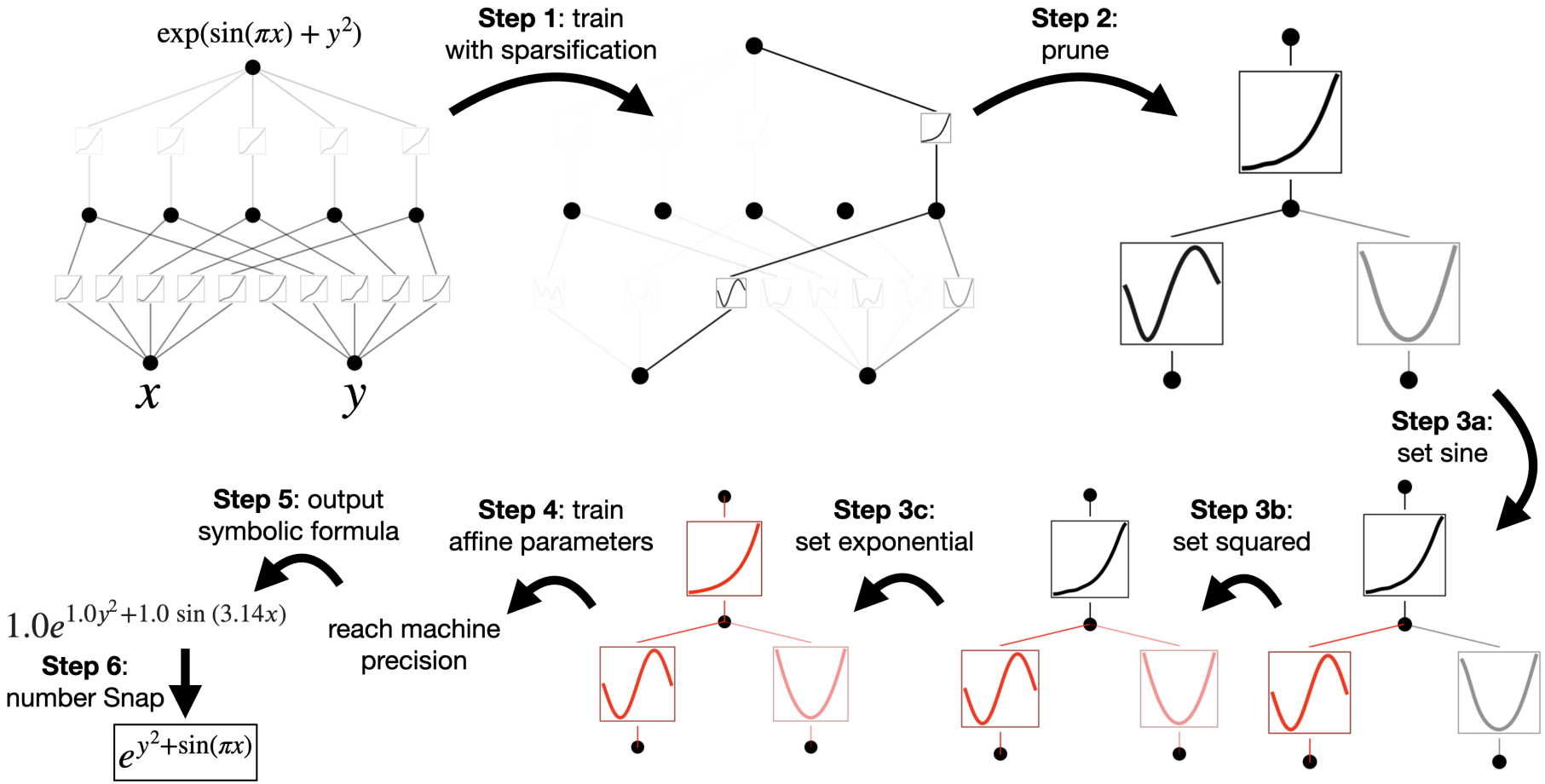


Figure 7: Regularization to important nodes improves interpretability.

Pruning: Automatic pruning is seen to discard all hidden neurons except the last one, leaving a [2, 1, 1] KAN. The activation functions appear to be known symbolic functions.

Continual Learning and Catastrophic Forgetting

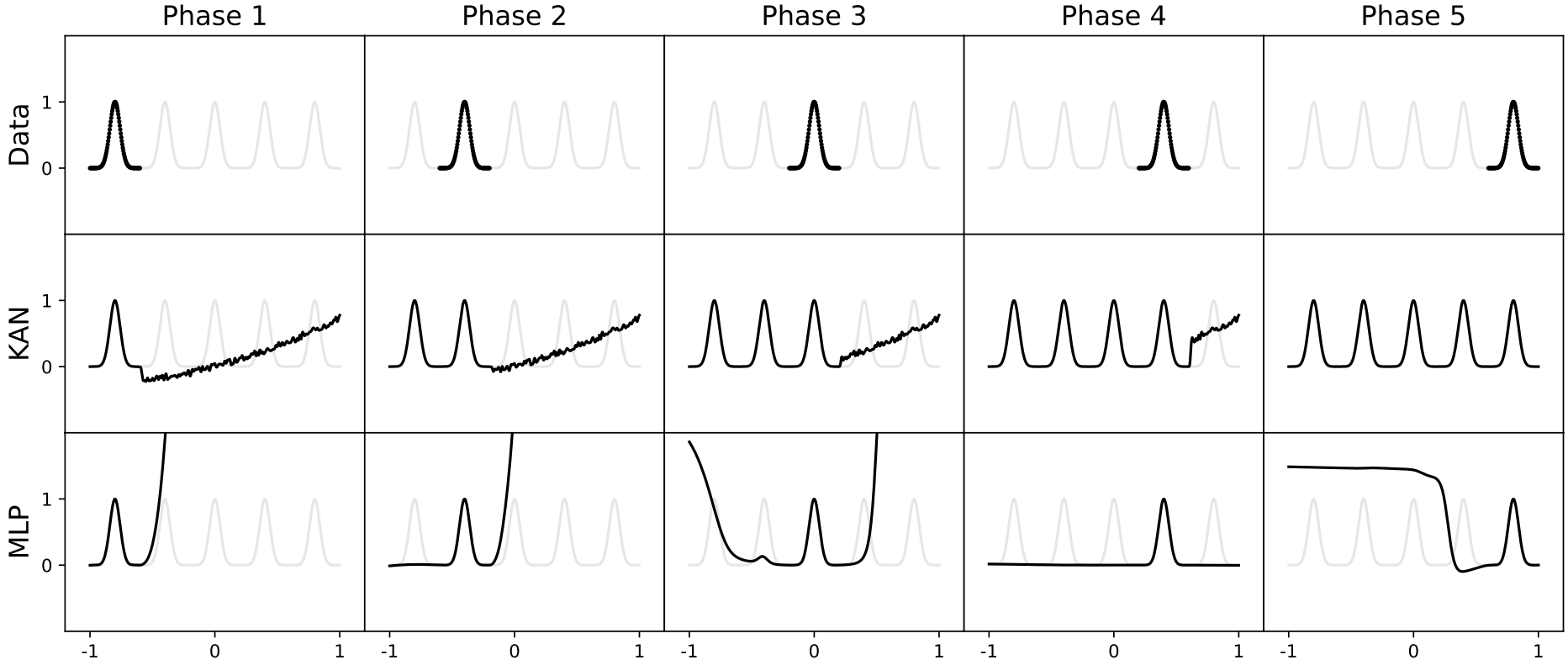


Figure 8: Learning the splines has local plasticity!

Spline bases are local, so a sample will only affect a few nearby spline coefficients. By contrast, since MLPs use global activations, any local change may propagate uncontrollably to regions far away, destroying the information being stored there.

Does KAN scale?

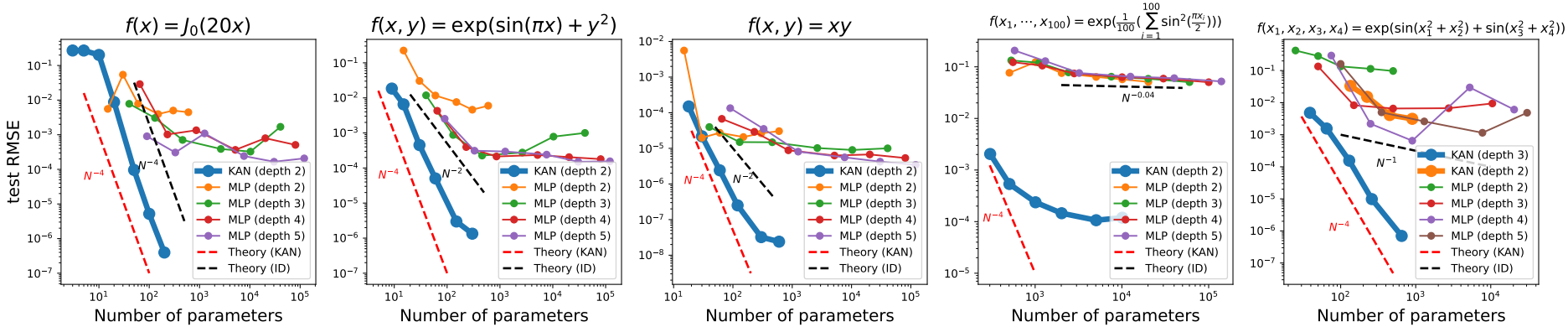


Figure 9: Compare KANs to MLPs on five toy examples. KANs can almost saturate the fastest scaling law predicted by our theory ($\alpha = 4$), while MLPs scales slowly and plateau quickly.

Section 4

Discussion