Interpretable AI for Scientific Discovery

Fabian Ruehle (Northeastern University & IAIFI)

Quantum 100 x AI Workshop - Münster
11/13/2025

Based on:

[Liu, Wang, Vaidya, FR, Halverson, Soljačić, Hou, Tegmark: 2404.19756]

[Harvey, FR, Fraser-Taliente, Halverson: 2505.07956]

[Gukov, Halverson, FR: 2402.13321]

AI in Science

- Despite its tremendous success, AI for science has shortcomings
- Neural Networks are
  - Black-box, i.e., neural networks are notoriously difficult to interpret
  - Stochastic, i.e., their output is probabilistic and depends on many (arbitrary) choices when setting up the algorithm
  - Error-prone, i.e., NN make mistakes, and it can be very hard to quantify or even detect them
- However, scientific discovery often necessitates
  - Understandable / interpretable results
  - Provably correct results (for mathematical proofs) with zero (not just quantifiable) error

Outline

Understandable/interpretable results (KANs)

- Use a different architecture with
  - Fewer parameters
  - Less polysemantic neurons
  - More straightforwardly interpretable
- Combine with symbolic regression

Outline

Understandable/interpretable results (KANs)

- Use a different architecture with
  - Fewer parameters
  - Less polysemantic neurons
  - More straightforwardly interpretable
- Combine with symbolic regression

Provably correct results (RL)

- Give up on interpreting the NN
- Solve (classification) task with RL
  - When learning Chess moves from Magnus Carlsen, we don't try to understand his brain, but the sequence of moves he played
  - Study episodic rollouts to show that result is correct and to learn the strategy that the algorithm discovered

Outline

Understandable/interpretable results (KANs)

- Use a different architecture with
  - Fewer parameters
  - Less polysemantic neurons
  - More straightforwardly interpretable
- Combine with symbolic regression

Provably correct results (RL)

- Give up on interpreting the NN
- Solve (classification) task with RL
  - When learning Chess moves from Magnus Carlsen, we don't try to understand his brain, but the sequence of moves he played
  - Study episodic rollouts to show that result is correct and to learn the strategy that the algorithm discovered

Honorable Mentions

Proof (Auto-) Formalization

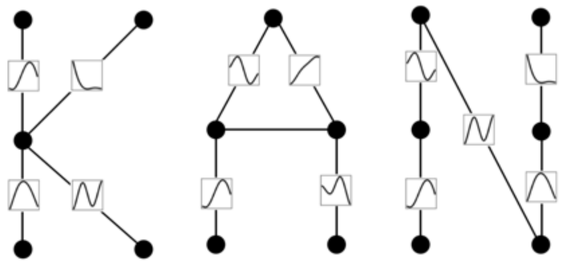
- Recently, formal languages like LEAN have sparked interest in the Math community, e.g. the Liquid Tensor Experiment of Scholze or Tao's formalization of mathematical proofs
- LEAN is powerful but difficult to learn (dependent type theory), but agentic LLMs are becoming better at writing proofs in LEAN

ML Theory

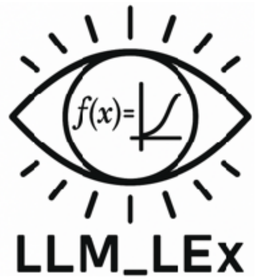
- Use theoretical results from ML to give guarantees about the NN performance
- An example is the NN-FT correspondence, which makes NNs tractable asymptotically
- This often leads to quantifiable errors, not zero errors

Outline

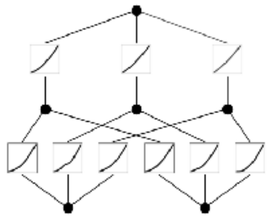
1 Kolmogorov Arnold Networks

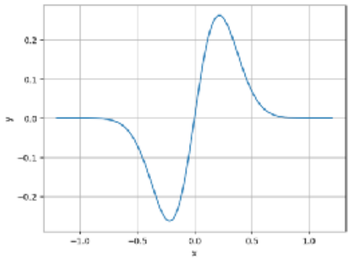


2 Symbolic Regression

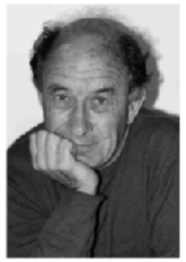


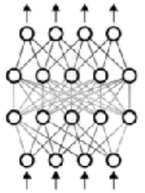
3 Conclusions

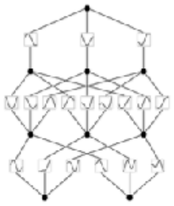




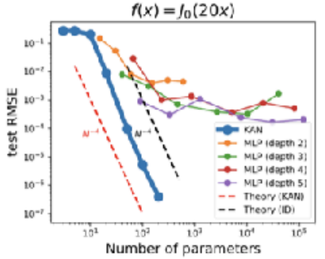
Kolmogorov + Arnold + Network = KAN

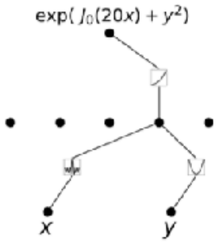


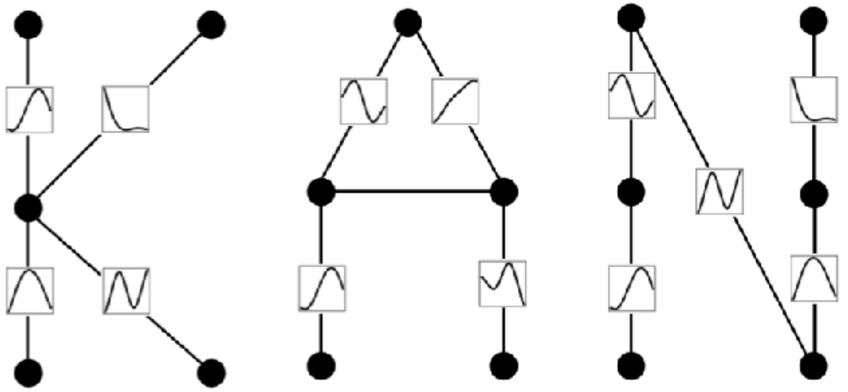




KANs are both **accurate** & **interpretable** !







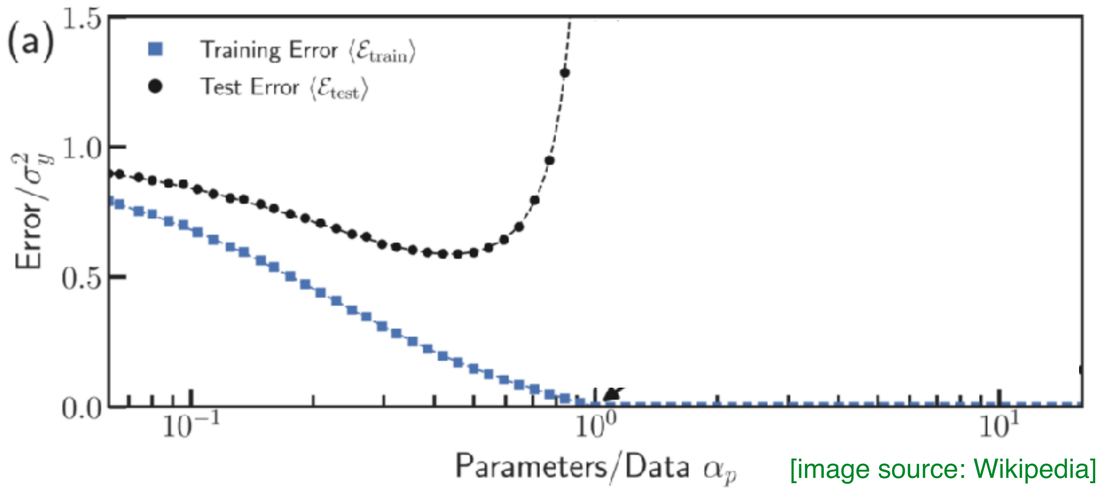
Mathematical
- Kolmogorov-Arnold Theorem — ***Section 2.1***
- KAN scaling laws — ***Section 2.3***

Accurate
- Methodology: Grid extension — ***Section 2.4***
- Application: Data fitting, PDE — ***Section 3***

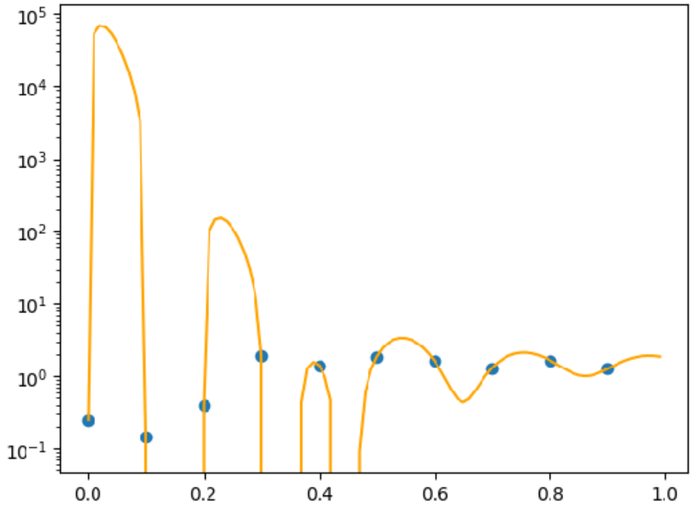
Interpretable
- Methodology: Simplification — ***Section 2.5***
- Application: AI for math & physics — ***Section 4***

Kolmogorov-Arnold Networks

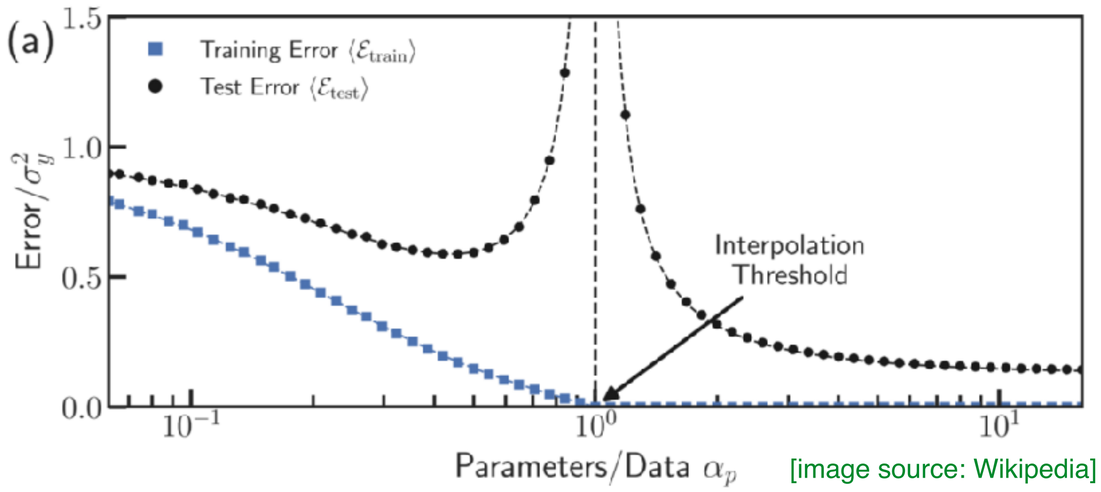
Neural Scaling Laws



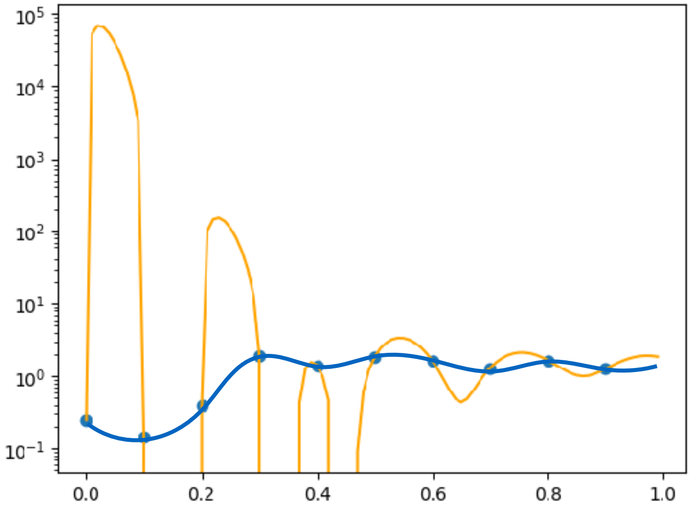
[image source: Wikipedia]



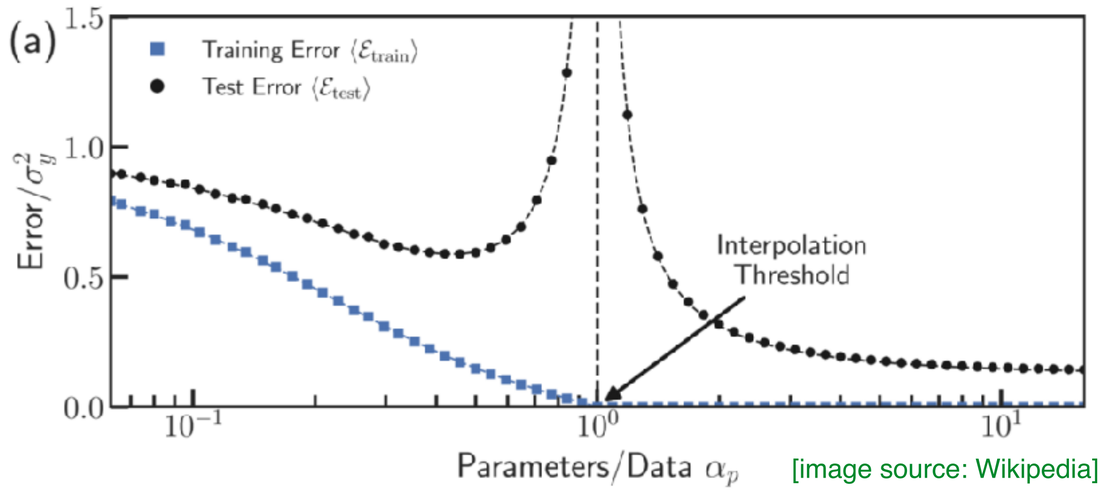
Neural Scaling Laws



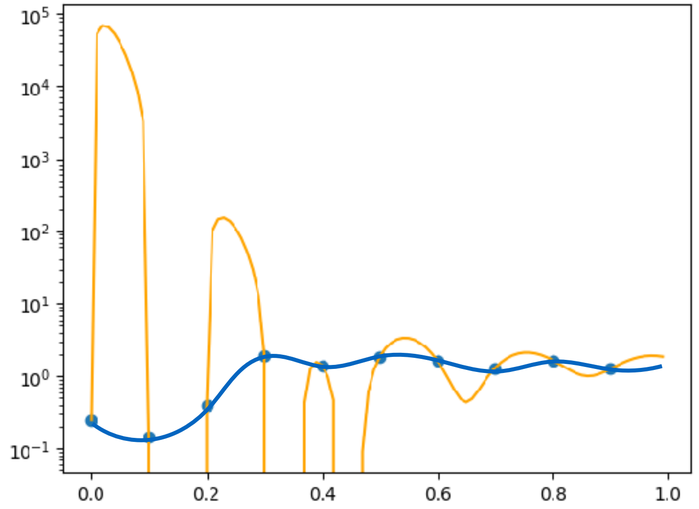
[image source: Wikipedia]

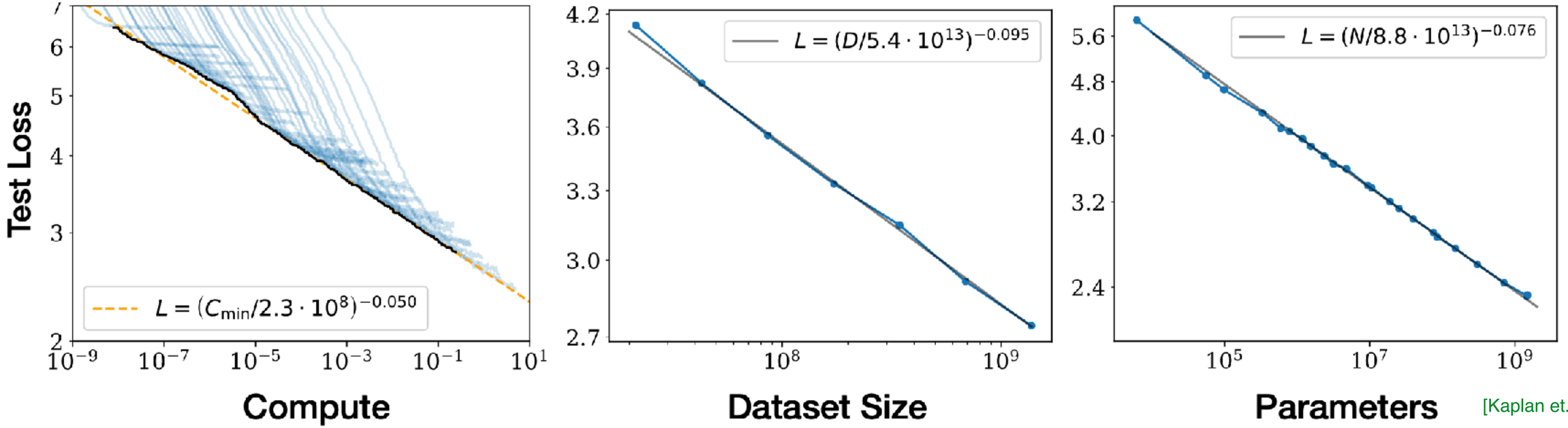


Neural Scaling Laws



[image source: Wikipedia]

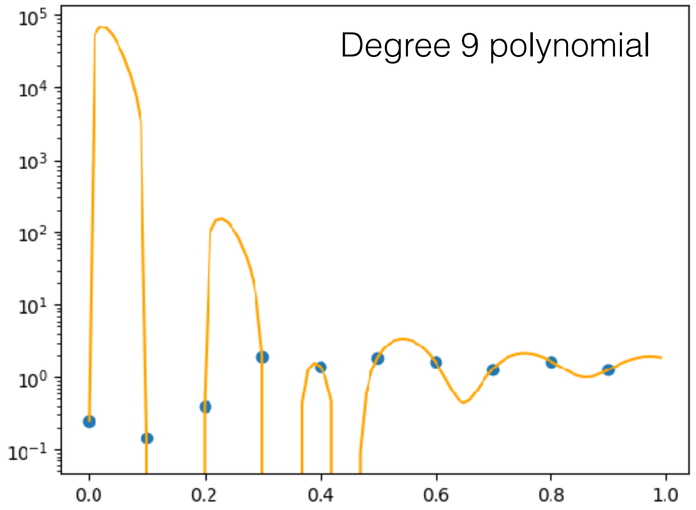


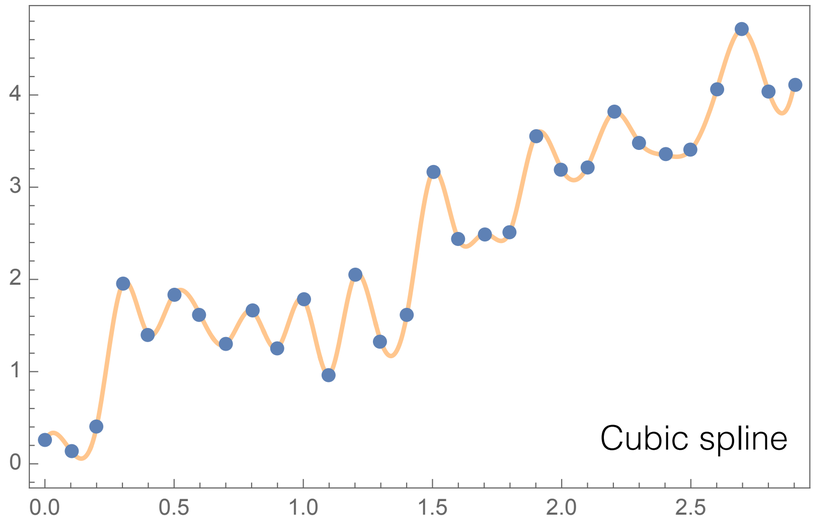


[Kaplan et.al. `20]

Fitting Functions - Splines

- Instead of fitting a single degree N polynomial to N+1 points, we can fit piecewise defined polynomials (splines)
- For example for a cubic spline $ax^3 + bx^2 + cx + d$, we have 4 parameters that are adjusted such that
  - The spline goes through the points
  - The derivatives at the points are smooth

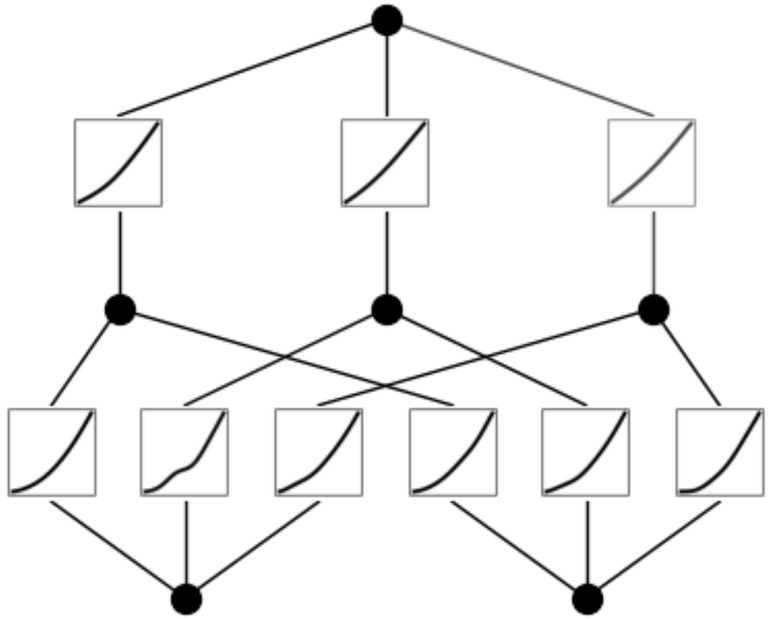




Learning Functions with KANs

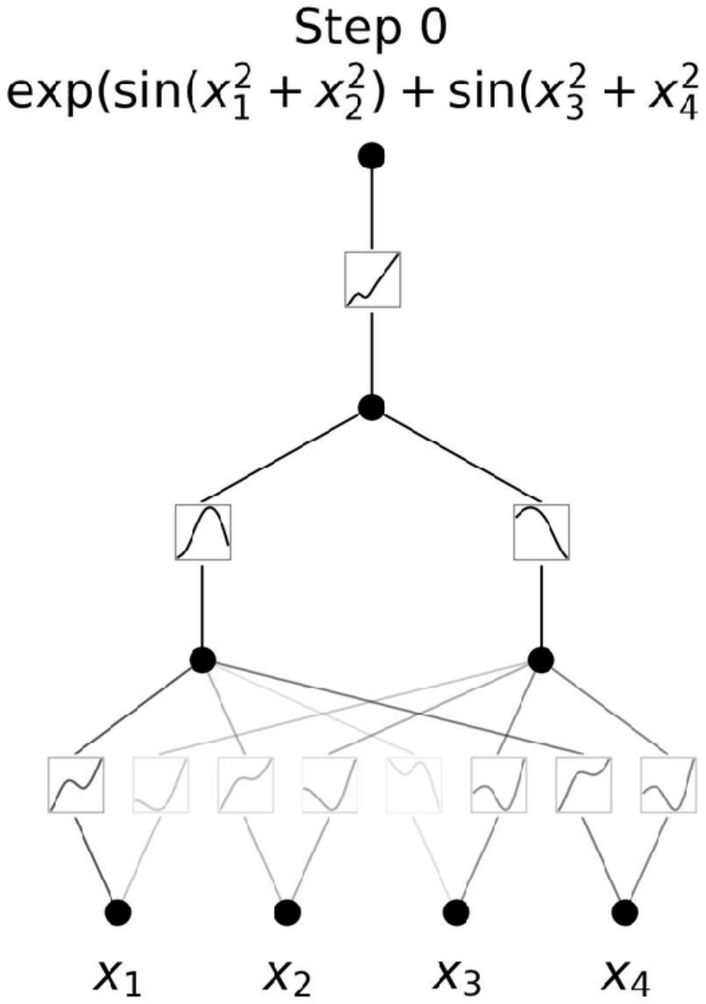
- A KAN parameterizes a function by alternating between
  - Single-variable functions (on edges): $\vec{x}_i \mapsto \sigma_j(\vec{x}_i)$
  - linear maps/additions (on nodes): $\vec{x}_i \mapsto \sum_i \sigma_j(\vec{x}_i)$
- In the example on the right we have a map $f : \mathbb{R}^2 \to \mathbb{R}$ expressed as

$$\mathrm{KAN}: \ \mathbb{R}^2 \xrightarrow{\sigma_j(\cdot)} \mathbb{R}^3 \xrightarrow{\sum_i} \mathbb{R}^3 \xrightarrow{\sigma_k(\cdot)} \mathbb{R} \xrightarrow{\sum_k} \mathbb{R}$$

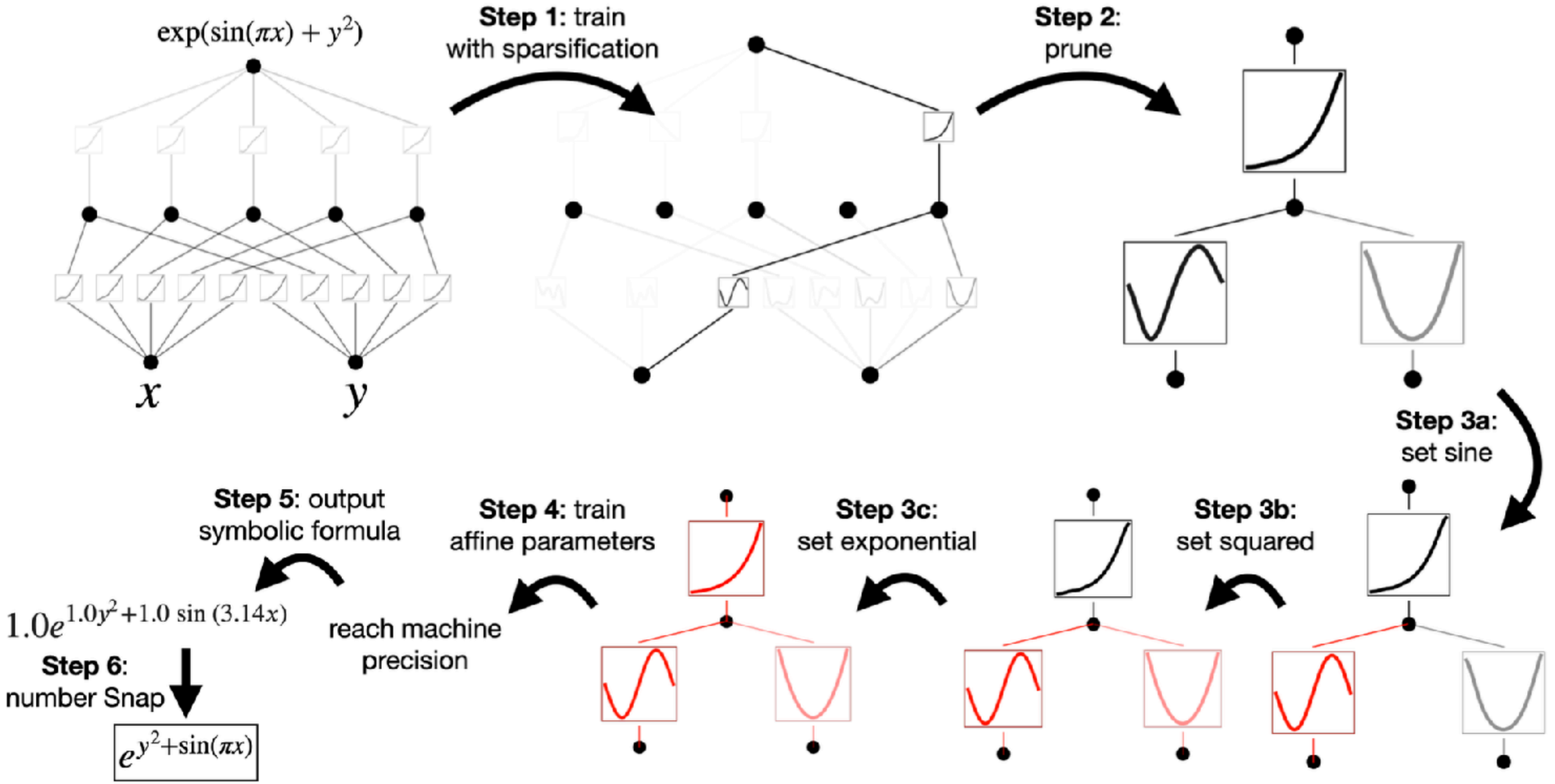


- The Kolmogorov-Arnold Theorem guarantees us that we can express any function we want to learn in this way

Example: Training a KAN



Interpretability of KANs - Workflow

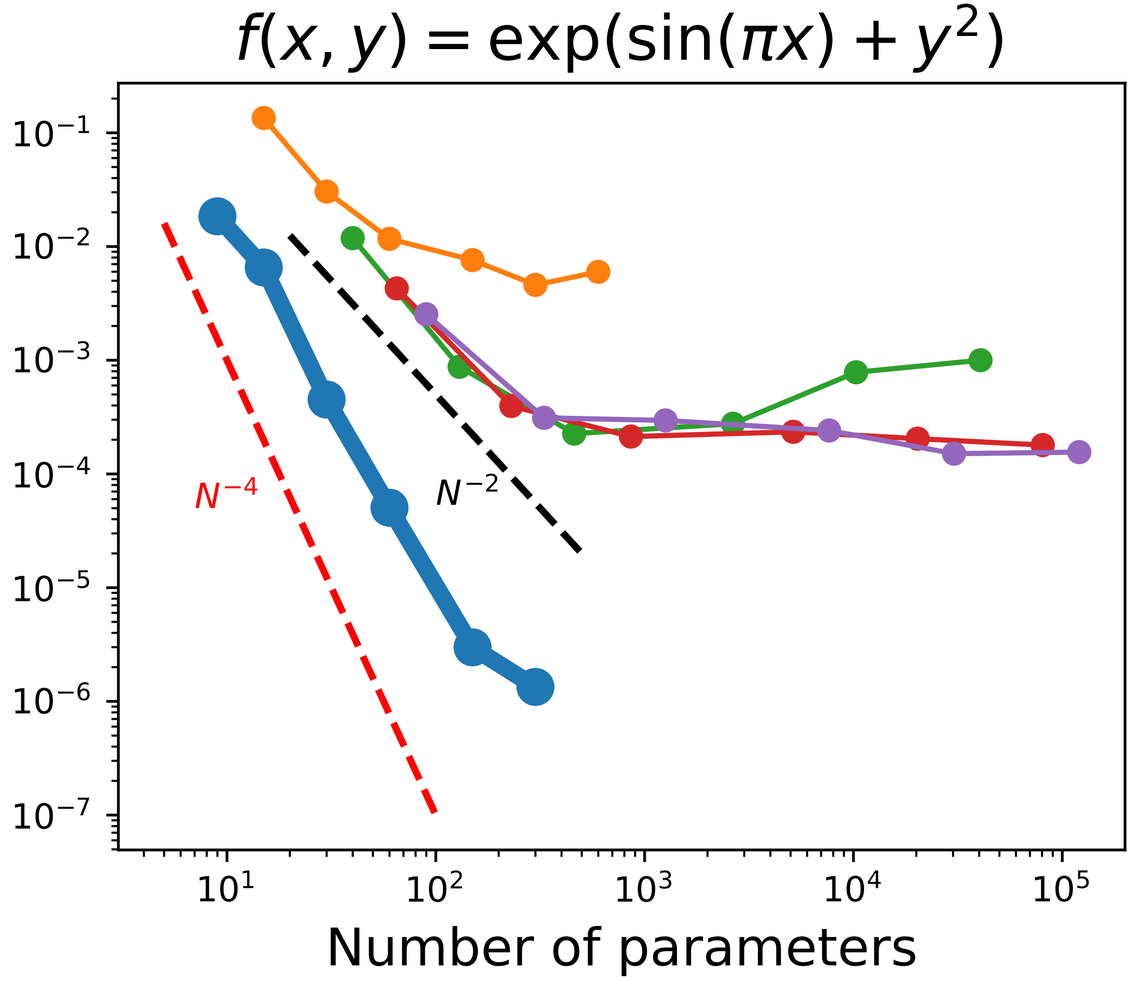


Neural Scaling Laws for KANs

- Neural scaling laws study how the test RMSE scales as a function of the number of parameters, training time, or training sample size (we focused on scaling with the number of parameters N): $\mathcal{L} \sim N^{-\alpha}$
- Kaplan&Sharma suggested that there is an intrinsic dimensionality of the input data manifold, and fitting piecewise polynomials of degree k (so ReLU would be $k = 1$) gives a fitting error of $\alpha = (k+1)/d$
- This suffers from the curse of (intrinsic) dimensionality
- For KANs, the Kolmogorov-Arnold theorem for piecewise polynomials of degree k gives a theoretic scaling of $\alpha = (k+1)$, defeating the curse of dimensionality! For cubic splines we hence expect $\alpha = 4$

Examples for Neural Scaling Laws - 2

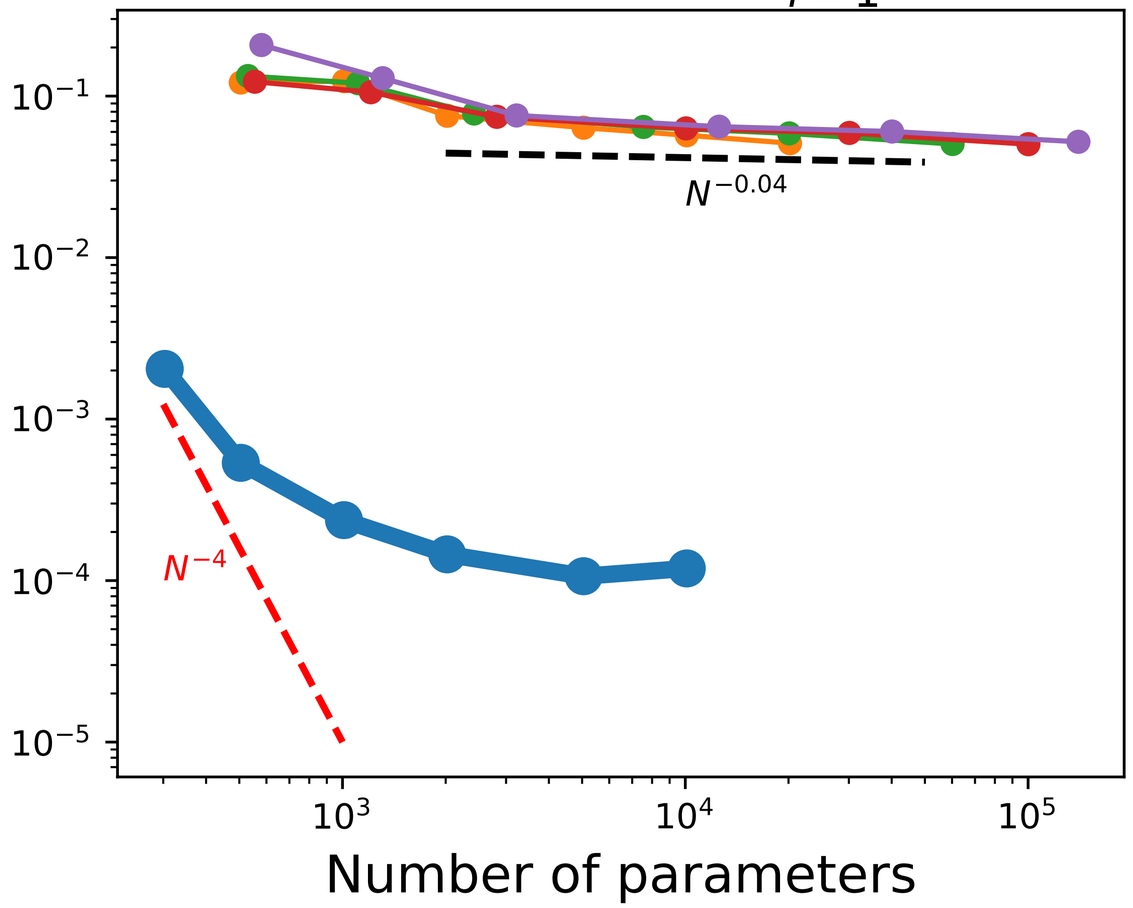
- As a second example, we study a composite function
  $f : \mathbb{R}^2 \to \mathbb{R}$
- This can be represented by a [2,1,1] KAN (learning sin and squared, adding the results, and taking exp), so we thus expect an N^{-4} scaling for cubic splines
- For NNs we expect an N^{-2} scaling, since the intrinsic dimensionality is 2



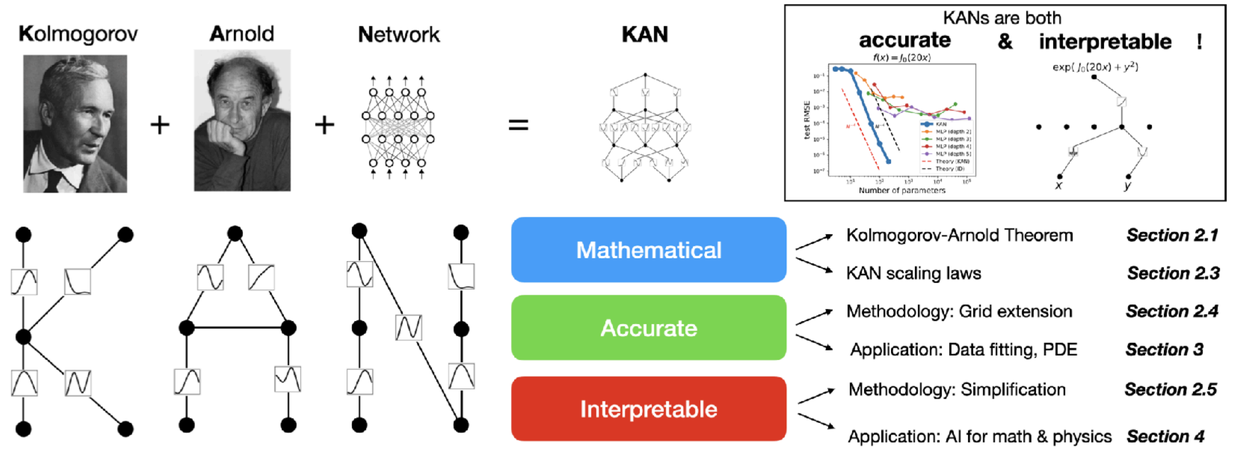
Examples for Neural Scaling Laws - 3

- As a high-dimensional example, we study a composite function $f : \mathbb{R}^{100} \to \mathbb{R}$
- This can be represented by a [100,1,1] KAN, so we thus expect an N^{-4} scaling for cubic splines
- For NNs we also an $N^{-2/100}$ scaling, since the intrinsic dimensionality is 100

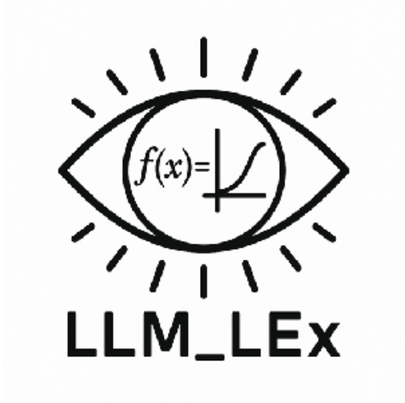
$$f(x_1, \cdots, x_{100}) = \exp\left(\frac{1}{100}\left(\sum_{i=1}^{100} \sin^2\left(\frac{\pi x_i}{2}\right)\right)\right)$$

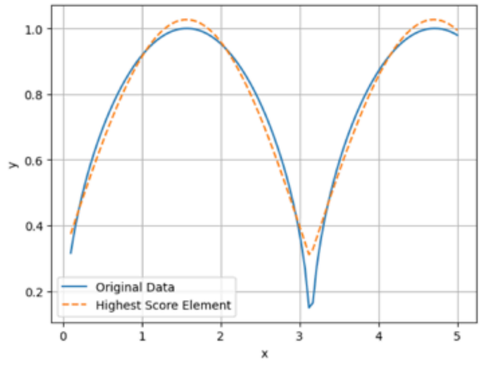


Try it yourself

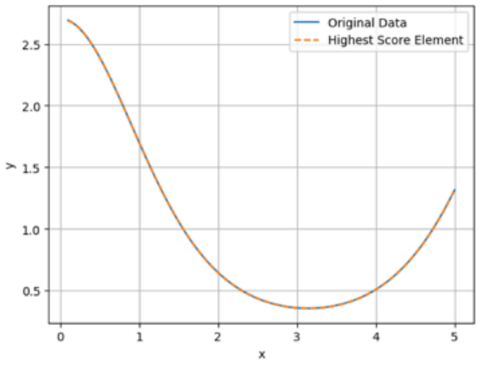


[https://github.com/KindXiaoming/pykan]

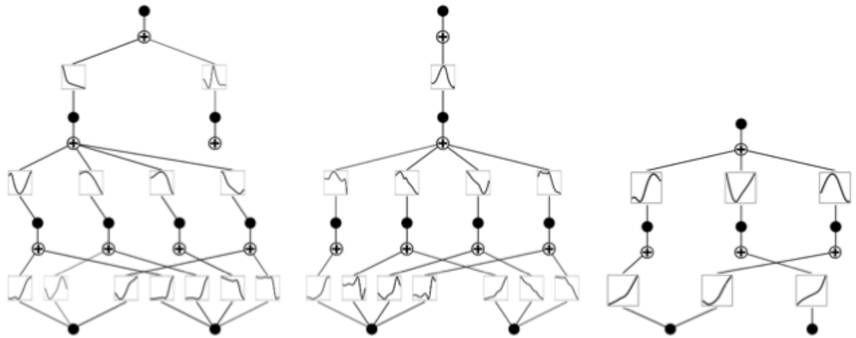




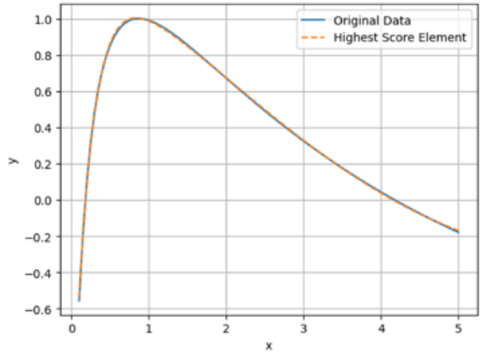
(a) Target Function: $\sqrt{|\sin(x)|}$



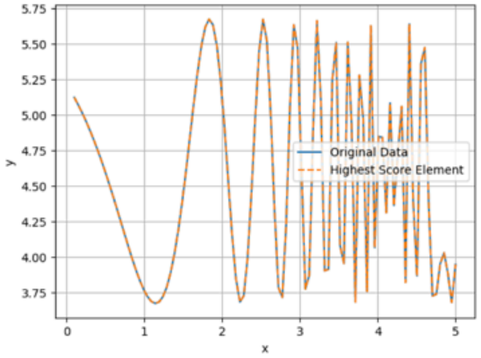
(b) Target Function: $e^{\cos(x)} - 0.0126997$



(a) $[2, 4, 4, 1]$ (pruned) (b) $[2, 4, 1, 1]$ (pruned) (c) $[2, 4, 1]$ (pruned) (d) $[2, 3, 1]$ (pruned)



(c) Target Function: $\sin\left(\log\left(\frac{4.1746}{x}\right)\right)$



(d) Target Function: $\cos(e^x) + 4.67315$

Large Lange Models Learning Expressions (LLM-LEx)

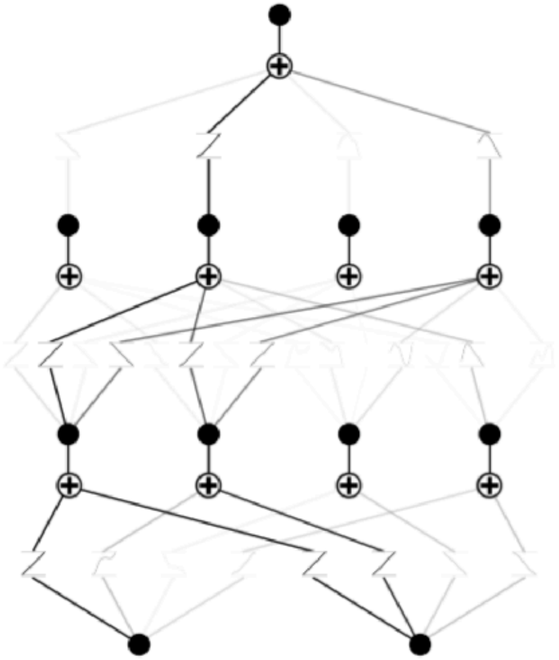
Univariance is all you need

- Use the KA representation theorem to write

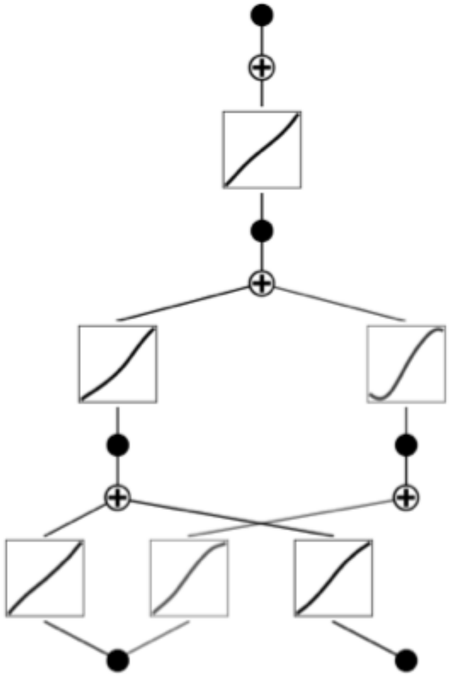
$$f(x_1, x_2, \ldots, x_n) = F(\sum_{i=1}^{n} f_i(x_i))$$

- Guess a (too big) ansatz for the architecture; prune and refine until you have reached a good starting point
- The individual f_i are univariate function, so we can plot them
- Feed each to a vision transformer and ask for an ansatz
- Use a GA (akin to fun-search) to iteratively improve the ansatze
- Fit the parameters in each ansatz
- Simplify the resulting function globally
- Can also prime the network (“The following dataset is from cosmology”)

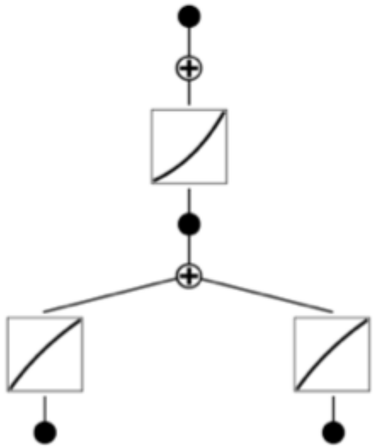
Examples



[2, 4, 4, 1]

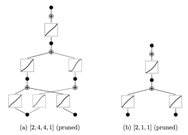


[2, 4, 4, 1] pruned



[2, 1, 1]

Examples

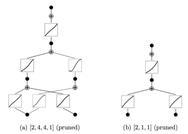
1. Guess an ansatz and prune 

2. Extract the splines as graphs, give them to ChatGPT, ask for parameterized ansatze, fit the parameters and score them. Repeat over multiple stages to improve the ansatze

$$
\begin{aligned}
e_{11}^{(1)}(x,y) &= a_0 \log(x) + a_1 \,, & \vec{a} &\approx (1.182, 0.261) \\
e_{21}^{(1)}(x,y) &= b_0 \log(y) + b_1 \,, & \vec{b} &\approx (1.182, 0.339) \\
e_{11}^{(2)}(z) &= c_0 \exp(c_1 z) \,, & \vec{c} &\approx (0.602, 0.846)
\end{aligned}
$$

Examples

1. Guess an ansatz and prune



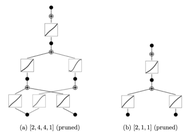
2. Extract the splines as graphs, give them to ChatGPT, ask for parameterized ansatze, fit the parameters and score them. Repeat over multiple stages to improve the ansatze

$$
\begin{aligned}
e_{11}^{(1)}(x,y) &= a_0 \log(x) + a_1 \,, & \vec{a} &\approx (1.182, 0.261) \\
e_{21}^{(1)}(x,y) &= b_0 \log(y) + b_1 \,, & \vec{b} &\approx (1.182, 0.339) \\
e_{11}^{(2)}(z) &= c_0 \exp(c_1 z) \,, & \vec{c} &\approx (0.602, 0.846)
\end{aligned}
$$

3. Global simplify the expression

$$
f(x,y) = c_0 e^{c_1(a_0\log(x)+b_0\log(y))+c_1(a_0+b_0)} \approx c_0 e^{c_1(a_0+b_0)} e^{\log(x)+\log(y)} \approx e^{\log(x)+\log(y)}
$$

Examples

1. Guess an ansatz and prune 

2. Extract the splines as graphs, give them to ChatGPT, ask for parameterized ansatze, fit the parameters and score them. Repeat over multiple stages to improve the ansatze

$$
\begin{aligned}
e_{11}^{(1)}(x,y) &= a_0 \log(x) + a_1 \,, & \vec{a} &\approx (1.182, 0.261) \\
e_{21}^{(1)}(x,y) &= b_0 \log(y) + b_1 \,, & \vec{b} &\approx (1.182, 0.339) \\
e_{11}^{(2)}(z) &= c_0 \exp(c_1 z) \,, & \vec{c} &\approx (0.602, 0.846)
\end{aligned}
$$

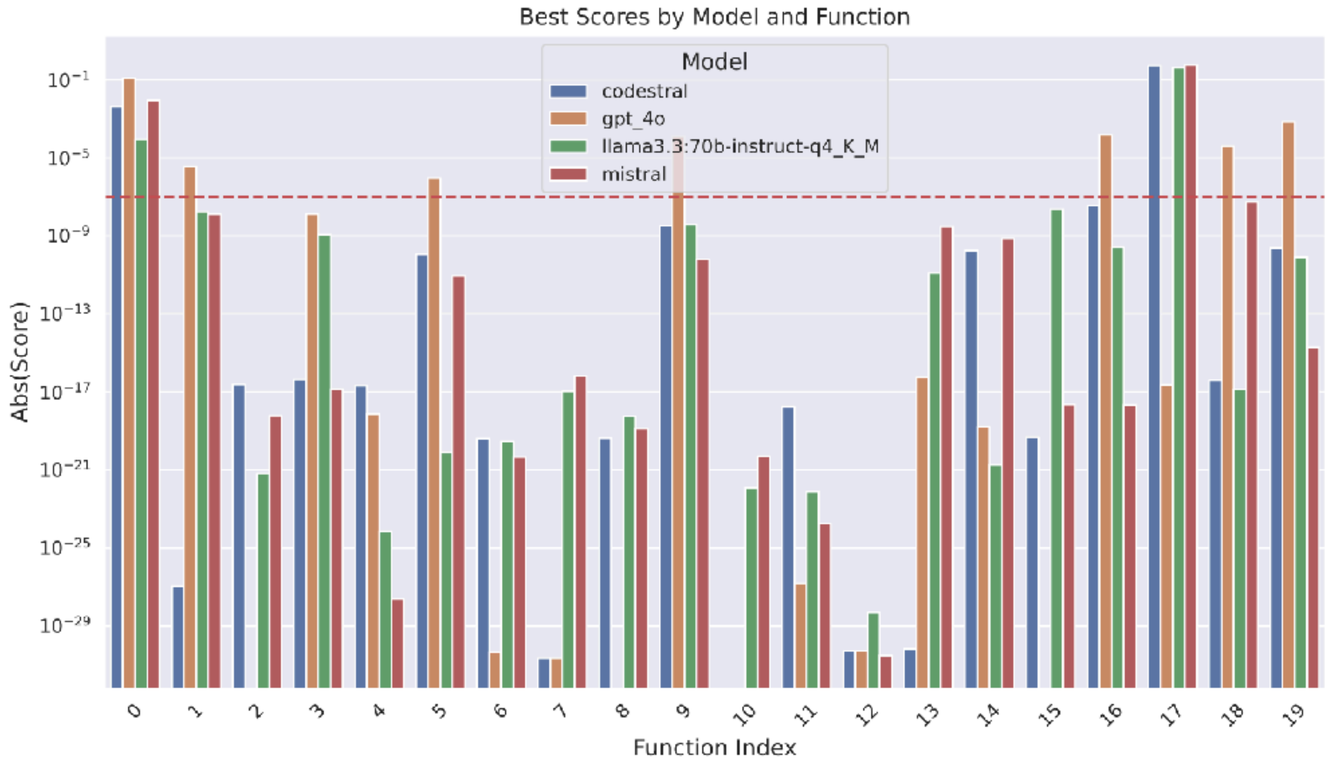
3. Global simplify the expression

$$
f(x,y) = c_0 e^{c_1(a_0 \log(x) + b_0 \log(y)) + c_1(a_0 + b_0)} \approx c_0 e^{c_1(a_0+b_0)} e^{\log(x)+\log(y)} \approx e^{\log(x)+\log(y)}
$$

4. Give this function to ChatGPT and ask for further simplifications

```
['x * y', 'y * x', 'x * y + 0', 'x*y', 'y*x','x**1 * y**1','x*y', 'x**1 * y**1', 'x*y + 0']
```

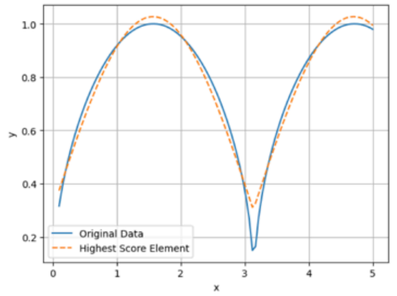
Open LLMs



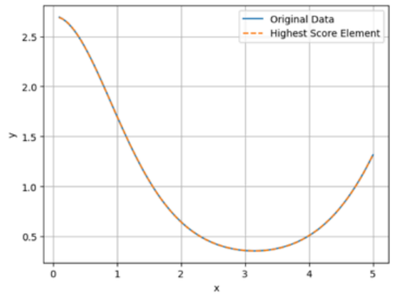
Expression	Mathematica	gpt-4o	llama3.3	mistral	codestral
$\sqrt{\lvert\sin(x)\rvert}$					
$e^{1.83169-\frac{3.35509}{x}}$					✓
x^3	✓	✓	✓	✓	✓
$(\sqrt{x}+1.44439)\,(\log(x)+\pi)$					
$3.09529x^3$	✓	✓	✓	✓	✓
$(x^3+\pi)^2$	✓				
$51.2288\cos(1.18219x)$		✓	✓		✓
$-55.0512\,(\sqrt{x}+1.)$	✓	✓		✓	✓
x	✓	✓	✓	✓	✓
$e^{\cos(x)}-0.0126997$					
$1.54251-x$	✓	✓	✓	✓	✓
e^{2x}	✓	✓	✓	✓	✓
$4.01209+e^x$		✓		✓	✓
$0.729202\sqrt{x}-\pi$		✓			✓
$-3x^3+x+1.99594$	✓	✓	✓		
$\log(x+1)$		✓		✓	✓
$\sin\left(\log\left(\frac{4.1746}{x}\right)\right)$					
$\cos\left(e^x\right)+4.67315$		✓			
$2e^{-3x}+e^{-x}$			✓		✓
$\frac{x+4.11509}{x^3}$					
Total Correct	8	12	8	8	12

Try it yourself

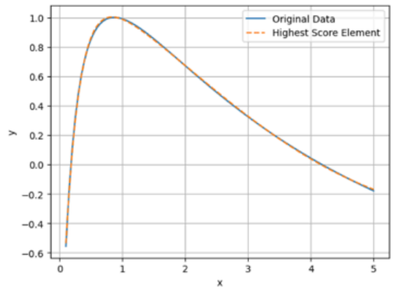
LLM_LEx



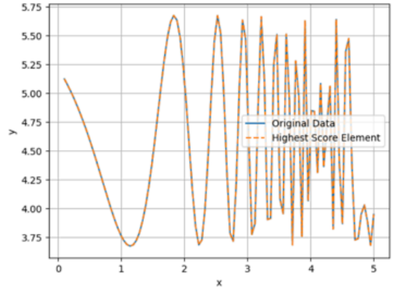
(a) Target Function: $\sqrt{|\sin(x)|}$



(b) Target Function: $e^{\cos(x)} - 0.0126997$



(c) Target Function: $\sin\left(\log\left(\frac{4.1746}{x}\right)\right)$



(d) Target Function: $\cos(e^x) + 4.67315$

[https://github.com/harveyThomas4692/llmlex]

Conclusions

1 KANs

- KANs are powerful alternative to NNs
- Sparsification, pruning, and visualization makes them more interpretable
- Univariance well-suited for multi-variate symbolic regression

2 Symbolic Regression

- Univariance well-suited for multi-variate symbolic regression
- Use FunSearch like Genetic Algorithm to “breed” better ansatze

Conclusions

1 KANs

- KANs are powerful alternative to NNs
- Sparsification, pruning, and visualization makes them more interpretable
- Univariance well-suited for multi-variate symbolic regression

2 Symbolic Regression

- Univariance well-suited for multi-variate symbolic reg
- Use FunSearch like Genetic Algorithm to “bre

Thank you!