Advanced Topic A15: *KAN vs MLP: Kolmogorov-Arnold Networks*

Data Science with Python – BSc Advanced Lectures

Joerg Osterrieder

10 Minutes

Motivation: Are Fixed Activation Functions a Limitation?

MLPs fix the activation function; KANs make it learnable

- The standard MLP: $h_{l+1} = \sigma(W_l h_l + b_l)$; activation σ is fixed (ReLU, GELU, sigmoid)
- The choice of activation is a hard architectural prior: the network can only compose instances of one function
- **Kolmogorov-Arnold Networks (KANs)** (Liu et al. 2024): place learnable activation functions on the *edges* instead of fixed activations on nodes
- Motivation: the Kolmogorov-Arnold representation theorem guarantees that any continuous function can be written as a composition of univariate functions
- KANs attracted significant attention in 2024 for symbolic regression and scientific discovery

KAN (Liu et al. 2024): activation functions are the parameters, not the weights between fixed activations

The Kolmogorov-Arnold Representation Theorem

Any continuous function is a sum of compositions of univariate functions

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

- Kolmogorov (1957): every continuous $f : [0,1]^n \to \mathbb{R}$ has this exact representation with $2n+1$ outer functions Φ_q and $n(2n+1)$ inner functions $\phi_{q,p}$
- The functions $\phi_{q,p}$ and Φ_q are univariate (one input, one output)
- Theoretical implication: any multivariate function can in principle be decomposed into layers of univariate operations
- Practical caveat: the original KA representation may require non-smooth inner functions; KANs use smooth splines as a practical approximation

KAT: the theoretical justification for placing all non-linearity on edges as univariate functions

KAN Architecture: Learnable Activations on Edges

Each edge carries a learnable spline function; nodes only sum

$$h_j^{(l+1)} = \sum_i \phi_{i,j}^{(l)}(h_i^{(l)}), \quad \phi(x) = w_b\, b(x) + w_s\, \text{spline}(x)$$

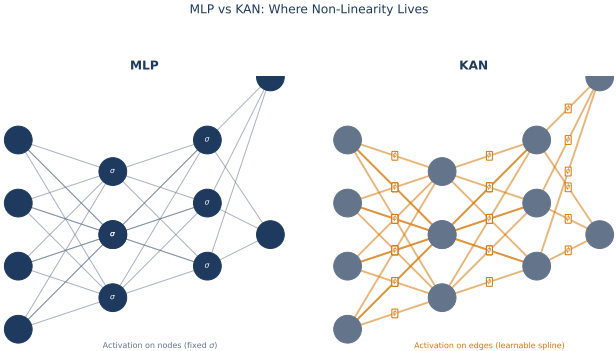
- Each $\phi_{i,j}$: a learnable B-spline parameterised by grid points and spline coefficients
- $b(x) = x/(1 + e^{-x})$: fixed SiLU basis; spline(x): learnable piecewise polynomial
- Nodes sum inputs without any non-linearity (unlike MLP nodes which apply σ after summation)
- The number of parameters scales as $O(n_{\text{in}} \cdot n_{\text{out}} \cdot G)$ where G is the grid size (spline resolution)

KAN inverts the MLP design: fixed linear edges and learnable node activations become learnable edge functions and fixed node summations

Architecture Comparison: MLP vs KAN

Different locations for non-linearity and different parameter types

- MLP: weights on edges (linear), fixed activations on nodes
- KAN: learnable spline activations on edges, sum-only nodes

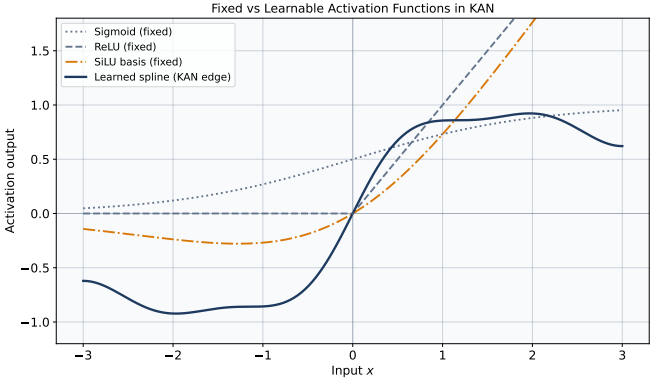


MLP: n fixed activations; KAN: $n_{\text{in}} \times n_{\text{out}}$ learnable spline activations

Learnable Spline Activations

Each edge learns a smooth function from its input data distribution

- B-splines: piecewise polynomial of degree k over G evenly-spaced grid points
- Grid is initially uniform; can be refined (adaptive grid extension) for better fit in complex regions
- The spline is differentiable: backpropagation through spline coefficients works exactly as through weights

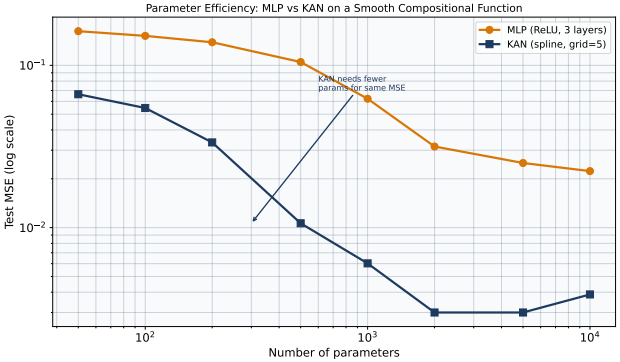


Spline activations adapt to the input distribution; fixed ReLU/GELU cannot

Parameter Efficiency: When Is KAN More Efficient?

KANs can achieve lower test error with fewer parameters for smooth low-dimensional functions

- For functions with sparse compositional structure (e.g., $f(x, y) = e^{\sin(\pi x)+y^2}$): KANs find the structure with fewer parameters than MLPs
- For unstructured high-dimensional data (images, language): MLPs and CNNs are more parameter-efficient
- The grid resolution G is a key hyperparameter: higher G gives more capacity but slower training



KANs win on smooth compositional functions; MLPs win on high-dimensional unstructured data

Symbolic Regression and Interpretability

KANs can discover symbolic formulas from data

- After training, each edge function $\phi_{i,j}$ can be visualised as a curve: if it looks like sin or x^2, the network has discovered that formula
- **Symbolic locking**: fix an edge to a specific symbolic function once identified; use the network as a symbolic regressor
- Liu et al. (2024): KANs rediscovered known physics equations (Feynman datasets) more accurately than MLPs
- **Pruning**: zero out small-magnitude edge functions; the remaining network is a sparse, interpretable graph
- Finance: a KAN trained on yield curve data may learn recognisable functions (Nelson-Siegel components) that a MLP would not reveal

KAN + symbolic locking: the network becomes a symbolic regression tool, not just a black-box predictor

KAN vs MLP: When Does KAN Win?

KANs have advantages for specific problem structures

- **KAN advantages**: smooth low-dimensional functions ($d < 20$); compositional structure; symbolic regression targets; scientific discovery tasks requiring interpretable equations
- **MLP advantages**: high-dimensional unstructured inputs; large-scale data; well-understood training dynamics; mature tooling and pre-trained models
- KAN training is slower per epoch: spline evaluation is more expensive than a matrix multiply
- For the same number of parameters: KAN often achieves lower approximation error on smooth functions
- On standard ML benchmarks (tabular, image, text): MLPs are competitive or better

KAN is not a universal MLP replacement: it is a specialist tool for smooth, interpretable, low-dimensional problems

Multi-Layer KAN and Composition

Stacking KAN layers builds deeper compositional representations

- A $[n_0, n_1, \ldots, n_L]$ KAN: L layers, each mapping n_l inputs to n_{l+1} outputs via $n_l \times n_{l+1}$ spline functions
- Deeper KANs compose more complex functions: $f = \Phi_L \circ \ldots \circ \Phi_1$
- Each layer can be visualised: the full composition is a directed graph of learned curves
- KAN width vs depth: shallow wide KANs vs deep narrow KANs; empirically similar; depth aids symbolic discovery
- Pruning after training: remove edges with near-zero magnitude; the remaining network is a sparse computational graph

KAN layers stack as function composition; the result is a differentiable, prunable computational graph

Training KANs: Practical Challenges

KANs require careful initialisation and grid management

- **Grid extension**: the initial uniform grid may not cover all input values; periodically extend and refine the grid as training data is seen
- **Spline initialisation**: small random coefficients; the SiLU basis ensures non-zero initial gradients
- **Regularisation**: L1 penalty on spline coefficients encourages sparse, interpretable functions; entropy regularisation promotes smooth activations
- **Longer training time**: on the same hardware, a KAN is 5–100x slower per epoch than an equivalent MLP due to spline evaluation overhead
- PyKAN (official library): automatic grid management; `kan.train()` handles all the above

KAN training needs grid extension and spline regularisation; PyKAN handles these automatically

Limitations of KANs

KANs are not a free upgrade over MLPs

- **Scaling**: no evidence that KANs scale to large models (billions of parameters); all strong results are on small networks
- **Training speed**: spline evaluation is not GPU-optimised to the level of matrix multiplications; practical throughput is much lower
- **Reproducibility**: initial KAN paper results required careful tuning; independent reproductions showed smaller gains
- **No pre-trained KANs**: the ecosystem of pre-trained models (LLMs, vision transformers) is built on MLPs; transfer learning is not available for KANs
- **High-dimensional limitation**: for $d > 20$ inputs, the number of edge functions grows as $d \times n_1$; the advantage over MLP evaporates

KAN 2024 hype was significant; the consensus in 2025 is: useful for scientific ML, not a general replacement

Recent Extensions: KAN 2.0 and Beyond

Active research extending KANs to new domains

- **KAN 2.0** (Liu et al. 2024b): multidimensional splines, deeper symbolic regression, improved training stability; introduced `MultKAN` for multiplicative interactions
- **Graph KAN**: replace the aggregation function in GNNs with a KAN; edge message functions are learnable splines
- **Temporal KAN**: replace LSTM/transformer layers with KAN layers for time-series; captures non-linear long-range patterns
- **Wav-KAN**: wavelet basis instead of B-splines; better for non-stationary signals (ECG, financial volatility)
- **Rational KAN**: rational functions instead of splines; faster evaluation, still learnable

KAN extensions are proliferating rapidly; the architecture is a research frontier rather than a production standard

KAN for Tabular Financial Data

Learnable activations reveal the functional form of each feature

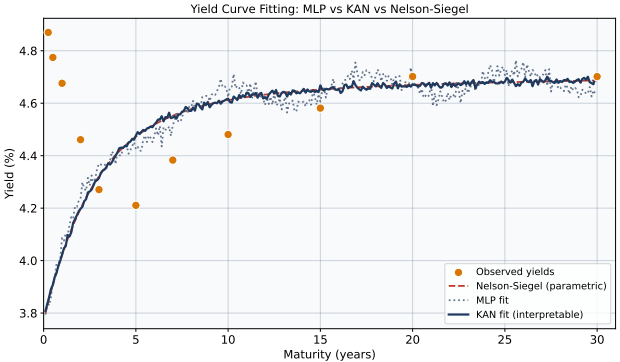
- Tabular ML benchmark: KAN competitive with XGBoost and CatBoost on small-to-medium structured datasets
- Each input feature gets its own learned edge function: the model explicitly shows whether the relationship is linear, logarithmic, or non-monotone
- **Monotonicity constraints**: financial relationships often must be monotone (more equity capital implies less default risk); KAN spline coefficients can be constrained to be monotonically increasing or decreasing
- **Credit scoring example**: KAN trained on 15 financial ratios; edge functions reveal a log-linear relationship for debt-to-equity and a sigmoid-like relationship for the current ratio
- Limitation: KAN does not natively handle categorical variables or missing values; preprocessing is still required before training

KAN on tabular data: interpretable functional forms per feature, not just feature importance scores

Finance Application: Yield Curve Fitting

Fitting a flexible term structure model with learnable activations

- Yield curve: map from maturity τ to yield $y(\tau)$; parametric models (Nelson-Siegel, Svensson) impose a functional form
- KAN approach: train a small KAN on observed yields; the learned edge functions reveal the functional form automatically
- If the edge function on a key node looks like $\exp(-\tau/\lambda)$: the network has discovered the Nelson-Siegel decay factor



KAN yield curve: the learned function is interpretable; the network can be locked to the discovered symbolic form

Finance Application: Option Pricing with KANs

Learning the Black-Scholes implied volatility surface

- The implied volatility surface $\sigma(K, T)$ maps strike K and maturity T to Black-Scholes implied volatility
- A KAN trained on observed option prices can approximate the surface and reveal its functional form
- Unlike a neural network black box: KAN edge functions can be inspected; a log-like function on K/S suggests a log-moneyness factor
- **Arbitrage constraints**: the IV surface must satisfy no-arbitrage conditions (calendar spread monotonicity, butterfly positivity); these are hard constraints that KAN does not automatically satisfy
- Hybrid approach: use KAN to discover the functional form, then fit a constrained parametric model

KAN for volatility surface: use it as a scientific discovery tool, not as an arbitrage-free pricer

Code Pointer

Kolmogorov-Arnold Networks in Python

- **PyKAN** (official): `pip install pykan`; `from kan import KAN`; supports symbolic locking, pruning, grid extension
- **Efficient KAN**: `pip install efficient-kan`; 10–100× faster implementation using `torch.nn.functional.linear`
- Basic training: `model = KAN(width=[2,5,1], grid=5, k=3); model.train(dataset, opt='LBFGS')`
- Symbolic regression: `model.prune(); model.suggest_symbolic(0, 0, 0)` to identify known functions
- Finance: wrap yield data as $([\tau], [y(\tau)])$ pairs; train a $[1, 3, 1]$ KAN; plot each edge function for symbolic discovery

efficient-kan is the production-speed implementation; pykan is for symbolic regression workflows

Summary and Open Questions

Three things to remember

- KANs place learnable B-spline activations on edges instead of fixed activations on nodes; the Kolmogorov-Arnold theorem provides the theoretical foundation
- KANs outperform MLPs on smooth low-dimensional functions and enable symbolic regression; on high-dimensional unstructured data, MLPs are more practical
- Finance applications: yield curve fitting and volatility surface modelling benefit from KAN's interpretability; prune and lock edge functions to discover symbolic parametric models

Open questions:

1. Can KANs scale to large models with billions of parameters?
2. How do we enforce financial no-arbitrage constraints directly in a KAN architecture?
3. Can temporal KANs replace transformers for long-horizon financial forecasting?

Further reading: Liu et al. (2024) KAN; Liu et al. (2024b) KAN 2.0; Bozorgasl & Chen (2024) Wav-KAN