Stochastic vs Quasi-Newton Optimization for Kolmogorov–Arnold Networks in Yield Curve Estimation

Zachary Olea

zolea1@jh.edu

Stochastic Optimization

Applied & Computational Mathematics

JHU Whiting School of Engineering EP

May 22, 2025

Abstract

Training Kolmogorov-Arnold Networks (KANs) for complex function approximation, such as modeling bond yield curves, presents significant optimization challenges due to highly nonconvex and potentially ill-conditioned loss landscapes, particularly when dealing with noisy financial data. KANs, inspired by the Kolmogorov-Arnold representation theorem, utilize learnable univariate activation functions to achieve both flexibility and interpretability, distinguishing them from standard neural networks. This work proposes a systematic framework to compare various stochastic first-order and zeroth-order optimization algorithms, including Stochastic Gradient Descent (SGD), variants of SGD, and Simultaneous Perturbation Stochastic Approximation (SPSA), against the quasi-Newton L-BFGS method used in the seminal KAN paper. This study aims to evaluate these algorithms in terms of computational scalability, robustness to noise and nondifferentiable behavior, and convergence properties in nonconvex settings. By identifying optimization techniques that effectively train KANs on noisy financial data, this research seeks to advance the state of yield curve estimation, balancing accuracy and interpretability for improved financial modeling.

---

1 Introduction & Motivation

Training Kolmogorov-Arnold Networks (KANs) for complex function approximation, such as modeling bond yield curves, poses a challenging optimization problem. KANs are neural networks inspired by the Kolmogorov-Arnold representation theorem, which guarantees that any continuous multivariate function can be represented as a finite composition and sum of univariate functions. Unlike standard multilayer perceptrons (MLPs) with fixed activations, KANs use learnable univariate activation functions, such as B-splines, enabling them to capture nonlinear dynamics while maintaining interpretability[1].

This flexibility, however, leads to highly nonconvex and potentially ill-conditioned loss landscapes, especially when fitting noisy financial data. To address this, I propose a framework to compare stochastic first- and zeroth-order optimization algorithms, Stochastic Gradient Descent (SGD) and Simultaneous Perturbation Stochastic Approximation (SPSA), against the quasi-Newton limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method used in the seminal KAN paper.

The motivation for using KANs in yield curve estimation is to balance accuracy and interpretability, a trade-off often encountered with traditional methods like splines and the Nelson-Siegel model. While L-BFGS was chosen in prior work for its straightforward implementation, the nonconvexity and noise in yield data suggest that stochastic optimization methods may offer improved robustness and convergence. By systematically comparing these algorithms, I aim to identify optimization techniques that can effectively train KANs on noisy financial data, enhancing yield curve estimation.

I will discuss the computational scalability, robustness to noise and nondifferentiable behavior, convergence properties in nonconvex settings, and practical implementation considerations for each algorithm. This research is crucial for advancing financial modeling and improving the accuracy of yield curve estimation.

2 Background

2.1 Traditional Methods

2.1.1 Yield Curves

In finance, yield curves show the relationship between interest rates and maturities for bonds[2]. Traditional estimation methods include parametric models like Nelson–Siegel (NS) and nonparametric approaches like splines. For a single yield curve, KAN is unnecessary, but if I want to model a panel of curves, KAN's multivariate structure becomes valuable. For example, I can model yield as $y(t,m)$, with t for time and m for maturity, letting KAN capture complex surfaces and latent factors through its sum-of-univariate structure. B-spline activations allow KAN to flexibly fit yield curve shapes, potentially outperforming the fixed form of NS.

2.1.2 Nelson-Siegel Model

The NS model is a widely used parametric approach for modeling yield curves, valued for its simplicity and interpretability. It describes the yield curve using three main factor: level, slope, and curvature, plus a decay parameter, capturing the general shape of the curve with just a few parameters. The model's functional form is:

$$y(t) = \beta_0 + \beta_1 \left(\frac{1 - e^{-\lambda t}}{\lambda t} \right) + \beta_2 \left(\frac{1 - e^{-\lambda t}}{\lambda t} - e^{-\lambda t} \right)$$

where β_0 is the long-term level, β_1 the short-term component, β_2 the medium-term component, and λ controls the exponential decay[3]. Extensions like the Svensson model add further terms to capture more complex shapes.

The NS model is popular in both academia and practice, especially among central banks, due to its ease of estimation and the economic interpretability of its parameters. However, its flexibility is limited; it may not capture unusual or highly irregular yield curve shapes.

2.1.3 Splines

Spline methods, especially cubic splines, offer a flexible, nonparametric alternative for yield curve modeling. Splines fit a smooth, piecewise polynomial curve to the data, ensuring continuity and smoothness at each knot. For n data points (x_i, y_i), the cubic spline $S(x)$ is defined as:

$$S(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \quad \text{for} \quad x_i \leq x \leq x_{i+1}$$

with coefficients a_i, b_i, c_i, d_i chosen to ensure smooth transitions between intervals. Splines can fit data very closely and are favored for their accuracy and smoothness, but they risk overfitting and their parameters lack direct economic interpretation. To address potential unrealistic shapes, variants such as exponential and maximum smoothness splines have been developed.

2.2 KAN

As proposed in the seminal paper[1], KANs leverage Kolmogorov's representation theorem by decomposing a multivariate function into sums and compositions of univariate functions. In a typical two-layer KAN, each input x_j passes through a learnable univariate function $\phi_j(x_j)$, and these outputs are linearly combined and passed through second-layer univariate functions $\psi_q(\cdot)$, which are then summed to produce the final output. Formally, for an n-dimensional input $\mathbf{x} = (x_1, \ldots, x_n)$,

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \psi_q \left(\sum_{j=1}^{n} a_{jq}\, \phi_j(x_j) + b_q \right),$$

where a_{jq} and b_q are trainable coefficients, and ϕ_j, ψ_q are learnable univariate functions. In practice, these univariate activations are often represented by flexible basis functions, such as B-splines. B-splines are piecewise polynomials with local support, allowing KANs to capture fine local features (e.g., humps or kinks in yield curves) while maintaining overall smoothness.

KANs bridge the gap between parametric and nonparametric approaches. Like splines, they are highly flexible and can approximate any continuous function given enough basis functions.

Their internal structure (sums of univariate functions) can, in principle, offer interpretability, as these components may correspond to latent factors.

KANs use more parameters than the Nelson-Siegel model, which increases the risk of overfitting if data is sparse, but they generally require fewer parameters than unconstrained splines if the Kolmogorov structure is enforced. For fitting a single yield curve, a spline may suffice, but KANs excel when modeling many curves simultaneously, capturing shared patterns across different contexts, such as throughout time, across countries, industries, and even firm-level. For example, I can train a KAN on historical data with inputs (date, maturity, etc.) and output yield, allowing it to learn time and maturity dependent components, similar to dynamic factor models.

In terms of fit, a well-trained KAN should outperform NS since it can capture non-linear relationships better and compared to splines, have the interpretable symbolic output from the b-splines. Robust optimizers, such as SPSA or SGD, and regularization can help prevent chasing outliers, while deterministic methods like L-BFGS may overfit unless explicit robustness is added.

A key challenge in training KANs is the highly nonconvex and potentially ill-conditioned loss landscape, due to the composition of nonlinear basis functions and the local influence of B-spline coefficients. This can result in many local minima, steep curvature, and flat or ill-conditioned directions, making optimization more complex than for standard neural networks. Gradient-based methods can struggle in these landscapes, so the choice of optimization algorithm is crucial. Advanced methods that account for curvature may converge faster, but only if they are robust to the irregularities of the loss surface.

3 Theoretical Considerations of Optimization Methods for KAN

3.1 Optimizers

3.1.1 L-BFGS

L-BFGS is a second-order quasi-Newton method that efficiently approximates the inverse Hessian using a limited history of past gradients, avoiding explicit Hessian computation. This makes it memory-efficient and scalable to high-dimensional problems, as it only stores a small number of n-dimensional vectors (with n the number of parameters)[4].

Each L-BFGS iteration requires computing the full gradient $\nabla f(\theta_k)$, followed by a series of vector operations proportional to the history length (typically a small constant, such as 5 or 10). For KANs, gradient computation via backpropagation is $O(n)$ per data sample, so a full gradient over N yield observations costs $O(N,n)$. The additional overhead for the Hessian update is $O(m,n)$, which is negligible if $m \ll N$ or n is moderate[1].

L-BFGS typically converges in fewer iterations than first-order methods, making it efficient even though each iteration processes the entire dataset. It is especially well-suited for high-dimensional spline networks, provided gradients are available. However, if the dataset is extremely large (N in the millions) or in streaming scenarios, L-BFGS becomes expensive due to the need for full-batch gradients. For most KAN applications—such as fitting a single yield curve or a moderate batch—N is manageable, and L-BFGS's per-iteration cost remains practical[1].

3.1.2 SGD

SGD is the standard optimizer for large-scale machine learning due to its ability to scale to massive datasets using mini-batches or single-sample updates. Each step computes a gradient $\nabla f(\theta_k;\xi)$ for a random mini-batch or single data point ξ, and updates parameters as $\theta_{k+1} = \theta_k - \eta_k \nabla f(\theta_k;\xi)$. This dramatically reduces per-iteration cost when N (the dataset size) is large, since the mini-batch size B is much smaller than N.

In yield curve fitting with small N, I might use the full batch, making SGD equivalent to

standard gradient descent. However, for training KANs on many yield curves or large datasets (across time or markets), SGD excels by decoupling computational cost from total data size. Each update costs $O(B,n)$, where n is the number of parameters, and $B \ll N$, making each step much cheaper than full-batch methods.

3.1.3 SPSA

SPSA is a stochastic, derivative-free optimization method ideal for situations where computing or storing exact gradients is difficult, or when the objective function is noisy. SPSA's key advantage is that it estimates the entire gradient using only two function evaluations per iteration, regardless of the parameter dimension n.

At each step, SPSA perturbs the current parameter vector in a random direction: it samples a random vector $\Delta_k \in \{\pm 1\}^n$ and evaluates the loss L at $\theta_k + c_k\Delta_k$ and $\theta_k - c_k\Delta_k$[5].The gradient is then approximated as:

$$g_k^{(\mathrm{SPSA})} \approx \frac{L(\theta_k + c_k\Delta_k) - L(\theta_k - c_k\Delta_k)}{2c_k}\,\Delta_k^{-1}$$

where the division by Δ_k is elementwise. This approach provides an unbiased gradient estimate under mild conditions.

The computational benefit is significant, even for thousands of parameters, SPSA requires only two forward passes through the KAN per iteration, compared to $2n$ evaluations for naive finite differences. Thus, SPSA's per-iteration cost is essentially constant with respect to n, making it scalable in parameter dimension. Memory requirements are also minimal, as only the current parameters and the perturbation vector need to be stored.

SPSA is especially attractive for complex KANs where analytic gradients are hard to derive, or when using external or legacy models. However, each iteration yields a noisy gradient estimate, so more iterations may be needed to reach convergence compared to exact-gradient methods. Still, when each function evaluation is expensive, such as when the KAN's loss involves costly simulations or market calibrations, SPSA's ability to minimize the number of evaluations is a major

advantage.

3.2 Noise and Nondifferentiability

I can see noise arising from measurement error in yield data and nondifferentiability may result from regularization terms (L^1 penalties) or from B-spline activations that are not perfectly smooth at knot points, though B-splines are usually at least C^1 continuous. Deterministic methods like L-BFGS assume a smooth, deterministic objective. L-BFGS relies on predictable loss decreases during line search and accurate curvature estimates from gradients[4]. When gradients are noisy, L-BFGS can mis-estimate curvature, leading to poor Hessian approximations or even divergence. L-BFGS also expects differentiability; at nondifferentiable points, its update formulas may not hold, making it less robust in such cases. This is why I believe SPSA and SGD may achieve better results than the standard L-BFGS implementation for my dataset.

3.3 Convergence

When training KANs, I am dealing with a nonconvex optimization problem, so none of these algorithms can guarantee finding the global optimum, only a local minimum or stationary point. However, their convergence rates and practical efficiency differ.

L-BFGS is known for fast local convergence, especially near a quadratic optimum. In convex quadratic cases, it achieves superlinear convergence, and even in nonconvex settings, it can converge rapidly if it finds a good basin of attraction. Empirically, L-BFGS often needs only tens of iterations on well-behaved problems, compared to hundreds for gradient descent. Its curvature approximation allows for larger, well-scaled steps, minimizing oscillations, especially useful if the loss landscape has steep and shallow directions. However, L-BFGS can get stuck at saddle points or poor local minima, depending on initialization, and may struggle in flat regions due to unreliable Hessian approximations[4].

For convex problems, SGD with a decaying learning rate achieves sublinear convergence, $O(1/\sqrt{t})$ or $O(1/t)$ for strongly convex cases. In nonconvex settings, it can only guarantee reach-

ing a stationary point, typically in $O(1/\varepsilon^2)$ iterations. SGD requires more iterations than second-order methods but each step is cheap. Momentum can help in ill-conditioned problems, but without adaptive step sizes, SGD may progress slowly in shallow directions. The stochasticity of SGD helps escape shallow local minima or saddle points, making it preferable in high-dimensional, non-convex problems. For KANs, SGD's noise can help explore multiple basins, while L-BFGS might commit to one.

SPSA behaves similarly to SGD, using stochastic gradient approximations. With decaying step sizes, it converges to a local optimum with probability 1, at a rate of $O(1/t)$ for mean squared error. Early on, SPSA may reduce loss more slowly due to noisy gradients, but it can escape shallow minima thanks to random perturbations. SPSA generally needs more iterations than SGD but is useful when exact gradients are unavailable.

4 Implementation

4.1 KAN

For my implementation, I employ B-spline activations due to their smoothness, interpretability, and ability to capture nonlinearity in a controlled fashion. Each layer in the network, denoted `KANLinear`, applies a combination of linear and spline-based transformations to its inputs. Formally, for an input $x \in \mathbb{R}^{d_{\text{in}}}$, the output of a KANLinear layer is given by:

$$y = W_{\text{base}}\,\sigma(x) + W_{\text{spline}}\,B(x), \tag{1}$$

where $W_{\text{base}} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ represents the base linear weights, $\sigma(\cdot)$ is a fixed base activation function (SiLU activation for this implementation), W_{spline} are the spline weights, and $B(x)$ denotes the B-spline basis expansion applied to the input features.

The B-spline expansion itself is constructed using a grid of knots, and basis functions $B_j(x)$ of order k are computed recursively. These basis functions allow the model to flexibly approxi-

mate nonlinear behaviors while maintaining locality, making them particularly suited to data with structured trends or discontinuities.

4.2 Objective and Regularization Training

I train the model by minimizing a regularized MSE objective function. The total loss function is defined as:

$$\mathscr{L}(\theta) = \mathscr{L}_{\text{MSE}}(\hat{y}, y) + \lambda_{\text{reg}} \mathscr{L}_{\text{reg}}(\theta), \tag{2}$$

where $\mathscr{L}_{\text{MSE}}$ is the standard mean squared error between predicted outputs $\hat{y}$ and true targets y, and $\mathscr{L}_{\text{reg}}$ is a regularization term applied to the spline weights.

The regularization component plays a dual role, penalizing both the overall magnitude of spline coefficients and their entropy. It is given by:

$$\mathscr{L}_{\text{reg}}(\theta) = \sum_{l} \left(\|\theta_{l,\text{spline}}\|_1 - \sum_{j} p_{l,j} \log p_{l,j} \right), \tag{3}$$

where $p_{l,j}$ represents the normalized magnitude of the j-th spline weight in the l-th layer. The first term enforces sparsity via an ℓ_1 norm, while the second term discourages overly diffuse activations, thus promoting interpretability and generalization.

4.3 Optimization Algorithms

To investigate the effect of different training algorithms on convergence behavior and generalization, I compare several first-order, one second-order, and one zeroth order optimization methods.

4.3.1 SGD

SGD updates parameters using gradients computed on mini-batches of data:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathscr{L}(\theta_k; \xi_k), \tag{4}$$

where η is the learning rate and ξ_k denotes the mini-batch at step k.

4.3.2 Adaptive First-order Methods

I also evaluate Adam, AdamW, RMSprop, and Adagrad, which dynamically adjust per-parameter learning rates based on past gradient magnitudes and variances. These optimizers have shown robustness in noisy and non-stationary optimization landscapes common in financial data.

4.3.3 Second Order: L-BFGS

As with the seminal paper, I tested L-BFGS, a quasi-Newton optimizer that approximates the inverse Hessian matrix using a limited memory of gradient history. The parameter update takes the form:

$$\theta_{k+1} = \theta_k - \eta H_k^{-1} \nabla_\theta \mathscr{L}(\theta_k), \tag{5}$$

where H_k^{-1} is the approximated inverse Hessian.

4.3.4 Zeroth Order: SPSA

As mentioned before, SPSA estimates the gradient using random perturbations in all dimensions simultaneously:

$$g_{\text{SPSA}}(\theta_k) = \frac{\mathscr{L}(\theta_k + c\Delta_k) - \mathscr{L}(\theta_k - c\Delta_k)}{2c\Delta_k}, \tag{6}$$

where Δ_k is a random perturbation vector and c is a small positive scalar. The update step is:

$$\theta_{k+1} = \theta_k - a\, g_{\text{SPSA}}(\theta_k), \tag{7}$$

with step size a scheduled to decay over iterations.

4.4 Training Procedure and Evaluation

At each training epoch, I record the mean squared error on the training, validation, and test sets; the regularization loss $\mathscr{R}(\theta)$; the total loss defined as $\mathscr{L}_{\text{total}} = \mathscr{L}_{\text{MSE}} + \lambda\mathscr{R}$; the gradient norm $\|\nabla_\theta \mathscr{L}\|$; the learning rate or step size η_t (when using adaptive methods); and the wall-clock time elapsed.

5 Empirical Results

I evaluated the models on the corporate bond yield prediction, tracking metrics including mean squared error on training, validation, and test sets, as well as the magnitude of regularization loss and gradient norms over epochs and computation time.

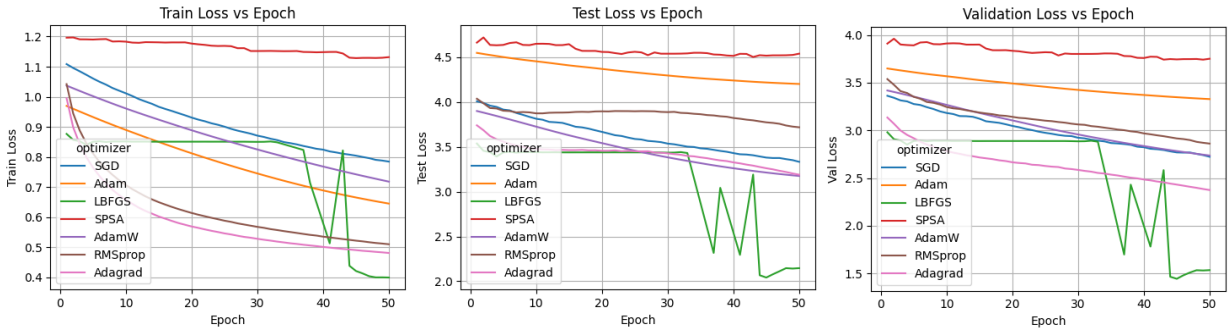


Figure 1: Results over Epoch

The results indicate that **L-BFGS** achieved the best performance in terms of final test loss on a per-batch basis, while AdamW and Adagrad achieved superior results when evaluated in terms of computational (wall-clock) time. Interestingly, while **RMSprop** performed comparably to **Adagrad** during training, its generalization to the test set was noticeably worse, a discrepancy that merits further investigation. Although **SPSA** underperformed in this setup, I still consider it a promising approach due to its low-cost, gradient-free updates and its robustness to noisy objectives. Its poor results here are likely attributable to limited hyperparameter tuning and an incomplete integration of the regularization term into the loss computation.

L-BFGS initially reduced the loss rapidly but eventually plateaued, likely due to entrapment in a local minimum or saddle point, an outcome consistent with the non-convex optimization landscape induced by spline-parameterized networks. At one point, both the MSE and regularization loss began to increase sharply, and I was not able to determine the root cause. Despite this instability, L-BFGS ultimately converged to the lowest training loss, albeit requiring significantly more wall-clock time. Based on this, I suspect that given equivalent compute time, the SGD-based methods would have achieved comparable performance.

Another observation concerns the test loss trajectory of Adagrad. Although I did not store model weights at each epoch, it appears that Adagrad initially reached the same suboptimal test loss plateau as L-BFGS, but escaped and improved more quickly than L-BFGS. This suggests that Adagrad may be more resilient to shallow traps in the optimization landscape. Finally, although train loss alone did not vary dramatically across optimizers, L-BFGS exhibited a considerably higher test loss, further highlighting the importance of evaluating performance under fixed computational budgets. That said, determining a consistent and principled training time budget across optimizers remains a challenge that I encountered during this experiment.

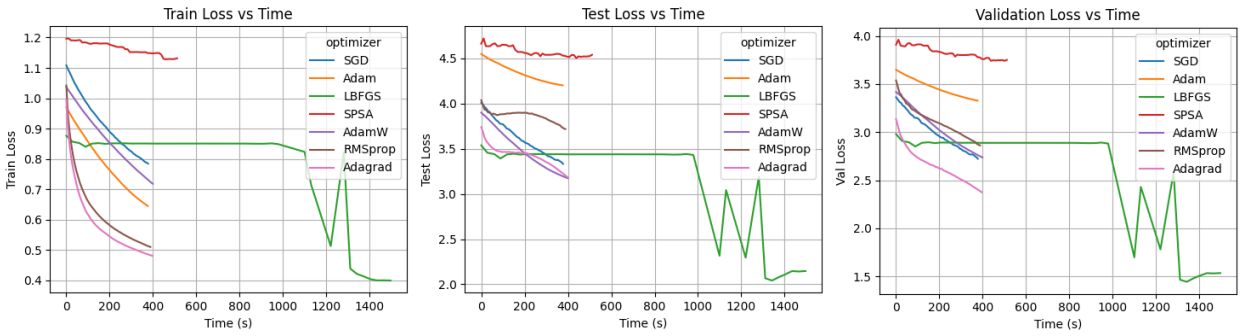
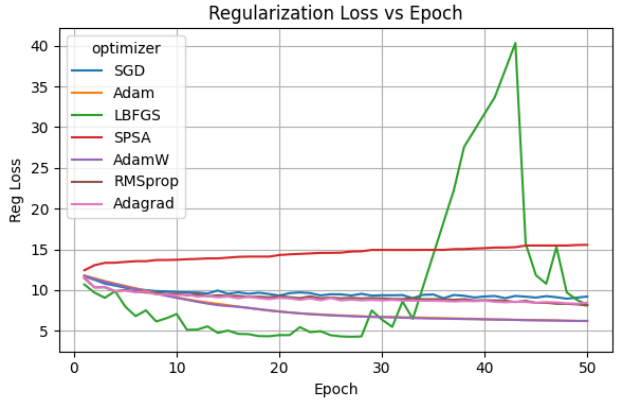


Figure 2: Results over Time

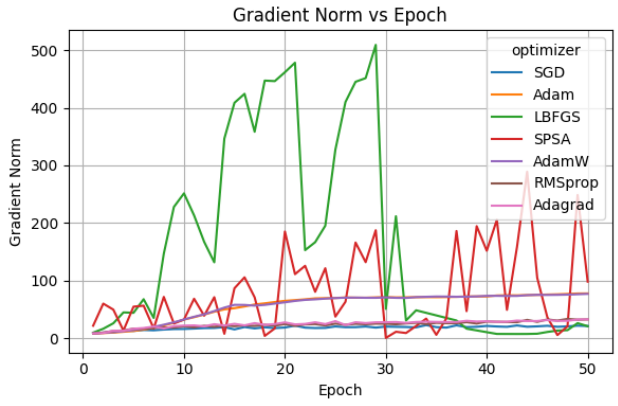
6 Analysis of Regularization and Gradient Norms

The behavior of the regularization term over time reveals important insights into model complexity. As expected, models that generalize well, such as with Adagrad, maintain lower regularization

loss, indicating simpler spline configurations and avoiding overfitting. In contrast, high regularization loss correlates with over-parameterization or unstable training. I did not properly implement the regularization loss with the SPSA method due to the model complexity and manual implementation.



Gradient norms also serve as a useful diagnostic tool. Large norms typically correspond to high curvature regions, while vanishing norms are indicative of flat regions or convergence basins. The use of gradient norm plots across epochs provided a valuable signal of optimizer performance and training dynamics. We can see L-BFGS has the largest until it explodes then reaches some convergence with the best train and test loss at the end of its training then flattens. SPSA's is the largest here and is indicative of the configuration for this study.



7 Conclusion and Future Work

The results of this study suggest that KANs can benefit from alternative optimizers rather than just L-BFGS for certain settings such as with noisy financial transaction data. My results also suggest that further research into KANs capabilities for bond yield curve modeling compared to splines or NS may be viable in that the models do converge and show improving test loss. Additionally, the current implementation omits macroeconomic variables, such as interest rate levels, inflation expectations, or central bank policy stance, which are standard inputs in most term structure models. Incorporating such covariates could improve generalization and reduce test loss, especially over longer horizons and under structural market shifts.

Looking ahead, a compelling extension would be to implement this framework in an online-learning setting. As new bond transaction data arrives, the model could update incrementally and provide real-time estimated yield curves. This would enable practitioners to assess whether a newly priced bond is trading at a discount or premium, and to make data-driven decisions on whether to buy, hold, or sell. In this context, optimizers like SGD and SPSA become particularly attractive. Both methods can perform fast, memory-efficient updates and do not require access to second-order curvature or large gradient histories, making them well-suited for streaming environments.

While SPSA was included in this study, its implementation lacked integration of the regularization loss due to time and code constraints. As a result, it was disadvantaged relative to other optimizers. However, had it included a properly weighted regularization term, along with an adaptive gain schedule (as seen in Adam and other SGD variants), it may have performed comparably. Unlike other methods, SPSA had to be implemented manually, while all other optimizers (e.g., L-BFGS, Adam, Adagrad) leveraged PyTorch's built-in functions[6]. Trying to manually adjust the KAN code architecture proved to be difficult but with careful implementation, could have promising results. Given the success of L-BFGS in the original KAN paper, there is reason to believe that second-order stochastic methods, such as 2SPSA and 2SGD, could offer robust convergence as well.

Finally, a promising future direction would be to adopt a hybrid optimization strategy. In par-

ticular, the KAN architecture separates naturally into linear weights and spline-based activations. One could update the linear parameters using SGD variants while applying SPSA to the spline coefficients, where gradient instability is more pronounced. Another hybrid approach would involve alternating between SPSA/SGD and L-BFGS during different phases of training, depending on the local curvature of the loss landscape. However, doing so would require careful management of optimizer state, especially for carrying forward gradient histories used by L-BFGS.

Overall, while KANs present optimization challenges, their functional flexibility justifies deeper investigation into tailored training strategies, particularly in high-noise or online learning contexts common in financial applications.

All experiments were implemented in PyTorch, and the full codebase, including cleaned bond transaction data and scripts for reproducing results, is available at `https://github.com/takeru240`.

A Results

A.1 Training after 50 Epochs

Optimizer Performance Summary (Part 1)

Optimizer	Epoch	Train Loss	Val Loss	Test Loss	Reg Loss	Total Loss
LBFGS	50	0.40	1.54	2.15	8.20	8.60
Adagrad	50	0.48	2.37	3.19	8.47	8.95
SGD	50	0.78	2.72	3.33	9.22	10.00
AdamW	50	0.72	2.74	3.17	6.23	6.95

Optimizer Performance Summary (Part 2)

Optimizer	Epoch	Train Loss	Val Loss	Test Loss	Reg Loss	Total Loss
RMSprop	50	0.51	2.86	3.72	8.12	8.63
Adam	50	0.65	3.33	4.20	6.21	6.85
SPSA	50	1.13	3.75	4.54	15.56	16.69

References

[1] Z. Liu, Y. Wang, S. Vaidya, *et al.*, *KAN: Kolmogorov–Arnold Networks*, arXiv preprint arXiv:2404.19756, Accepted to the International Conference on Learning Representations (ICLR) 2025, 2024. [Online]. Available: https://arxiv.org/abs/2404.19756.

[2] PIMCO, *Bonds 102: Understanding the yield curve*, https://www.pimco.com/us/en/resources/education/bonds-102-understanding-the-yield-curve, Accessed: 2025-05-02, 2024.

[3] C. R. Nelson and A. F. Siegel, "Parsimonious modeling of yield curves," *The Journal of Business*, vol. 60, no. 4, pp. 473–489, 1987.

[4] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989. DOI: 10.1007/BF01589116.

[5] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Hoboken, NJ: Wiley-Interscience, 2003, ISBN: 978-0471330523.

[6] PyTorch Contributors, *torch.optim — PyTorch Optimization Package*, Accessed: 2025-05-01, 2024. [Online]. Available: https://pytorch.org/docs/main/optim.html.