Tampere University

Nguyen Duc Minh

KOLMOGOROV-ARNOLD NETWORKS VS. MULTI-LAYER PERCEPTRONS

Bachelor's Thesis
Faculty of Engineering and
Natural Sciences
Jyrki Nummenmaa
April 2025

ABSTRACT

Nguyen Duc Minh: Kolmogorov-Arnold Networks vs. Multi-Layer Perceptrons

Bachelor of Science Thesis

Tampere University

Bachelor's Degree Program in Science and Engineering

April 2025

This thesis presents a comparative study of Kolmogorov-Arnold Networks (KANs) and Multi-Layer Perceptrons (MLPs), focusing on their architectural principles, learning behavior, and empirical performance. KANs are a recently proposed neural network model based on the Kolmogorov-Arnold representation theorem, which states that any continuous multivariate function can be decomposed into a composition and summation of univariate functions. Unlike MLPs, which rely on dense matrix multiplications and fixed activation functions, KANs apply learnable univariate spline-based transformations along each connection, enabling local and interpretable nonlinear modeling.

This study evaluates the two models on both synthetic and real-world tasks. A symbolic regression task is used to analyze model behavior in controlled settings, while a sentiment classification task using the IMDb dataset serves as a real-world benchmark. Experimental results show that KANs achieve lower or comparable test error with significantly fewer parameters compared to MLPs. Additionally, KANs trained with the L-BFGS optimizer exhibit better generalization and smoother learned functions. However, KANs incur higher training time due to spline evaluations and second-order optimization, especially in CPU-only environments.

The results demonstrate that KANs offer a promising alternative to traditional MLPs, particularly in applications where interpretability, parameter efficiency, and functional smoothness are desired. While further development is needed to improve scalability and training efficiency, the findings support the growing interest in functional and interpretable neural architectures.

Keywords: Kolmogorov-Arnold Representation, Kolmogorov-Arnold Networks, Multi-Layer Perceptrons, Function Approximation, B-Splines, Sentiment Classification, Neural Network Architecture, Parameter Efficiency.

USE OF AI IN THESIS

I have utilised AI tools in my thesis:

☐ No
☒ Yes

The AI tools utilised in my thesis and their purposes are described below:

Names and versions of AI tools: ChatGPT

Purpose of using AI tools:

- Reviewing and rephrasing technical content to improve clarity and academic tone.
- Providing idea and guidance on scientific writing and offering suggestions for organizing experiments.

Sections where AI tools were used: I used AI to summarize the theories and to polish my sentence when editing section 2, and to give me some idea to analyze the results in section 3.5 Results and Analysis.

I acknowledge that I am fully responsible for the entire content of my thesis, including the parts generated by AI, and accept accountability for any violations of ethical standards in publications.

PREFACE

I would like to express my deep gratitude to my supervisor, Professor Jyrki Nummenmaa, for giving me the opportunity to work as his research assistant, which has been a defining experience in my academic journey. His guidance, encouragement, and trust not only supported this thesis but also inspired me to pursue my future career path.

I also wish to thank my family and friends for their support throughout this process.

Tampere, 20 April 2025

Nguyen Duc Minh

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

MLP	Multi-Layer Perceptron
KAN	Kolmogorov-Arnold Network
ANN	Artificial Neural Network
MSE	Mean Square Error
SGD	Stochastic Gradient Descent

1. INTRODUCTION

In recent years, artificial neural networks have become the foundation of modern machine learning, demonstrating exceptional performance across a wide range of applications, including computer vision, natural language processing, and scientific modeling. Among them, the Multi-Layer Perceptron (MLP) has long served as a fundamental building block, leveraging stacked layers of linear transformations followed by nonlinear activations to approximate complex functions. Despite its success, the MLP architecture exhibits several limitations, particularly in terms of interpretability, parameter efficiency, and robustness in low-data or incremental learning settings.

To address these challenges, this thesis investigates Kolmogorov-Arnold Networks (KANs), a recently introduced neural architecture grounded in the Kolmogorov-Arnold representation theorem. Unlike MLPs, which rely on matrix multiplication and fixed nonlinearities, KANs construct function approximations by composing and summing trainable univariate functions along network edges. These functions are typically implemented using spline basis functions, allowing the model to learn smooth and interpretable transformations.

This study compares KANs and MLPs across both theoretical and practical dimensions. We begin by analyzing their respective mathematical foundations and structural characteristics. We then implement both architectures using PyTorch and evaluate them on two types of tasks: symbolic regression, using a handcrafted continuous function, and text classification, using the IMDb movie review dataset. We also investigate the behavior of KANs under continual learning conditions, where their localized parameter updates offer potential advantages over global architectures like MLPs.

The remainder of this thesis is structured as follows. Section 2 provides theoretical background on MLPs and the Kolmogorov-Arnold theorem, and introduces the architecture and training methods of KANs, including their spline-based representation and potential for continual learning. Section 3 presents the implementation details of both models, including dataset preparation, training setup, and the use of PyKAN. Section 4 summarizes the results and offers conclusions regarding the performance, interpretability, and efficiency of KANs compared to MLPs. The final section lists all references used throughout the study.

2. BACKGROUND

Artificial neural networks have become fundamental in machine learning, providing powerful tools for function approximation and pattern recognition. Among them, Multi-Layer Perceptron (MLP) and Kolmogorov-Arnold Network (KAN) represent two distinct approaches to learning complex functions.

MLPs rely on interconnected layers of neurons, utilizing activation functions and backpropagation to refine their predictions. They are widely used for classification, regression, and other machine learning tasks.

On the other hand, KAN is based on the Kolmogorov-Arnold Representation Theorem, which states that any multivariate function can be expressed as a composition of univariate functions and summations.

This approach challenges the conventional fully connected structure of MLPs by reducing reliance on matrix multiplications, offering a different paradigm for function approximation.

Understanding these two theoretical foundations, learning mechanisms, and computational efficiency is essential for assessing their advantages and limitations in various machine learning applications.

2.1 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP), a class of artificial neural networks (ANN) consisting of multiple layers of neurons organized feedforward.

MLP consists of three main components: an input layer that receives the input features and passes them to the next layer, hidden layers that perform computations using weighted connections and non-linear activation functions, and an output layer that produces the final prediction or classification result.

Each neuron in a layer of a MLP is connected to every neuron in the following layer, making it a fully connected network. These connections are parameterized by weights, which determine the influence of each input feature on the next layer. Bias terms allow shifting activation functions to improve flexibility in capturing complex patterns.

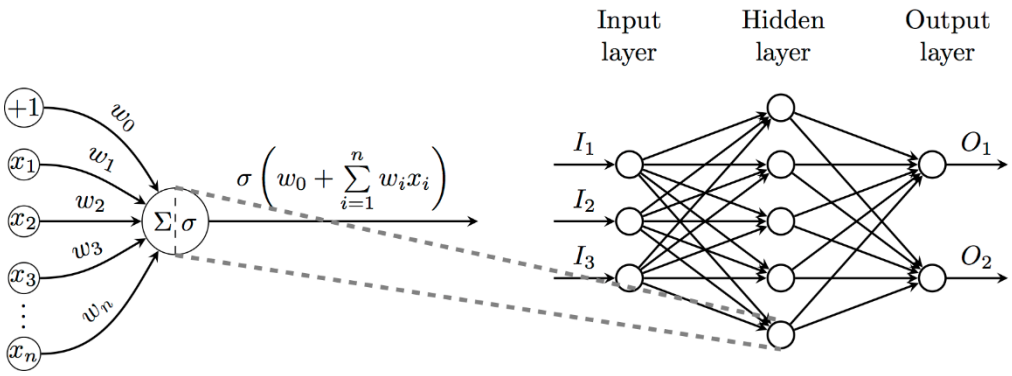

Figure 2.1. *Architecture of a MLP showing input, hidden and output layers.*

More specifically, each neuron in a MLP computes sum of the weighted inputs and then applies an activation function to produce a non-linear output, as shown in the figure 2.2.

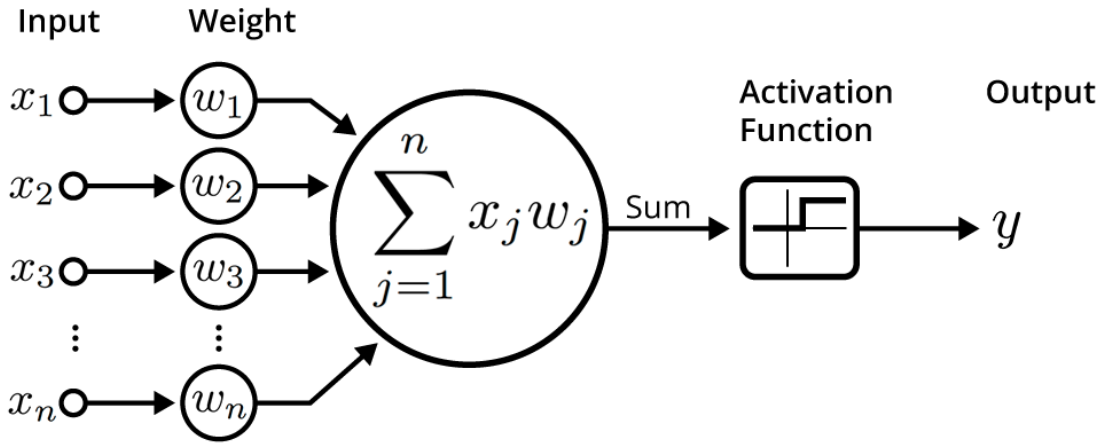

Figure 2.2. *A picture of an artificial neuron in a MLP, illustrating weighted inputs, summation, activation function and output.*

Mathematically, for a neuron receiving inputs $x_1, x_2, \ldots, x_n$ with corresponding weights $w_1, w_2, \ldots, w_n$. The neuron computes:

$$z = \sum_{j=1}^{n} x_j w_j + b \tag{2.1}$$

where b is the bias term. The activation function $\sigma(z)$ is then applied to produce the output.

This function-based transformation enables MLPs to approximate complex relationship in data, making them powerful universal approximators, as described by the Universal Approximation Theorem. This theorem states that any continuous function can be approximated by a MLP containing at least one hidden layer with sufficient neurons. This

means that MLPs do not rely on explicit rule-based programming but instead learn complex mappings through function approximation.

MLPs learn by adjusting their weights through backpropagation, a method that reduces the difference between predicted and actual results. The training process involves performing a forward pass to compute neuron activations from input to output, calculating the loss, and then performing a backward pass to compute gradients of the loss function with respect to the weights using the chain rules. Finally, the weights and biases are updated using gradient descents, where:

$$W^{(l)} = W^{(l)} - \eta \nabla L \tag{2.2}$$

with η is the learning rate and ∇L represents the gradient of the loss function with respect to the weights.

Common loss functions include Mean Squared Error (MSE) for regression [11]:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.3}$$

And Cross-Entropy Loss for classification:

$$L = -\sum_{i=1}^{N} y_i \log \hat{y}_i \tag{2.4}$$

Optimization techniques, such as Stochastic Gradient Descent (SGD) [2] and the Adam optimizer [3], are algorithms used to efficiently update network weights during training. Adam is particularly popular due to its adaptive learning rate capabilities and typically faster convergence compared to traditional SGD.

Activation functions introduces non-linearity, allowing them to model complex patterns. Common activation functions are the Sigmoid function, which is suitable for binary classification

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

And ReLU (Rectified Linear Unit) [1]:

$$f(x) = \max(0, x) \tag{2.6}$$

2.2 Kolmogorov-Arnold Representation Theorem

Kolmogorov-Arnold Networks (KAN) are inspired by the Kolmogorov-Arnold theorem, which provides a theoretical basis for approximating multivariate functions. Introduced by Andrey Kolmogorov and further developed by Vladimir Arnold, demonstrates that any continuous function of multiple variables can be represented through compositions and summations of univariate continuous functions [4].

The Kolmogorov-Arnold theorem states that for any continuous function on a bounded domain $f: [0,1]^n \to \mathbb{R}$, there exist continuous univariate function $\phi_{q,p}: [0,1] \to \mathbb{R}$ and $\Phi_q: \mathbb{R} \to \mathbb{R}$ such that:

$$f(x) = f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right) \tag{2.7}$$

Consider a multivariate continuous function defined as:

$$y = f(x_1, \dots, x_n)$$

According to the theorem, this multivariate function can be decomposed into a combination of single-variable functions. Specifically, we first define a set of univariate functions:

$$\phi_1(x_1), \phi_2(x_2), \dots, \phi_n(x_n) = \sum_{q=1}^{n} \phi_q(x_q)$$

Then, the output of these function is passed through another univariate function (single composition):

$$f \left(\sum_{q=1}^{n} \phi_q(x_q) \right)$$

To increase the approximation power, multiple compositions of these univariate functions can be utilized, such as m different compositions and sum them up [5].

$$\sum_{p=1}^{m} f_p \left(\sum_{q=1}^{n} \phi_{pq}(x_q) \right)$$

This approach effectively decomposes complex multidimensional relationships into simpler univariate transformations and combinations. KAN models thus utilize this mathematical insight to construct neural network architectures, focusing on univariate functions rather than traditional matrix-based neuron computations like in MLPs [6].

2.3 KAN Architecture

The KAN architecture follows the idea of the Kolmogorov-Arnold theorem and turns it into a simple neural network design. Instead of using traditional neurons that rely on matrix multiplications followed by activation functions (as in MLPs), KAN consists of layers where each connection is a learnable univariate function.

In KAN, each input dimension is independently processed through learnable univariate transformations, often parameterized as splines. These transformations act as the primary mechanism of learning, allowing the network to capture complex nonlinear behaviours. The outputs of these univariate functions are then passed on to the next layer. Importantly, the transformation does not occur within the nodes, but along the edges of the network. The nodes are responsible for summing the incoming transformed signals, which are then passed through the next set of functions. This is in contrast to traditional MLPs, where both transformation and summation happen inside the neurons.

More generally, the structure of a KAN can be viewed as a composition of layers of functional transformations:

$$\mathrm{KAN}(\mathrm{x}) = (\boldsymbol{\Phi}_{L-1} \circ \boldsymbol{\Phi}_{L-2} \circ \ldots \circ \boldsymbol{\Phi}_{0})\mathrm{x} \tag{2.8}$$

Where each layer $\boldsymbol{\Phi}_l$ applies a collection of univariate spline functions to its input vector components.

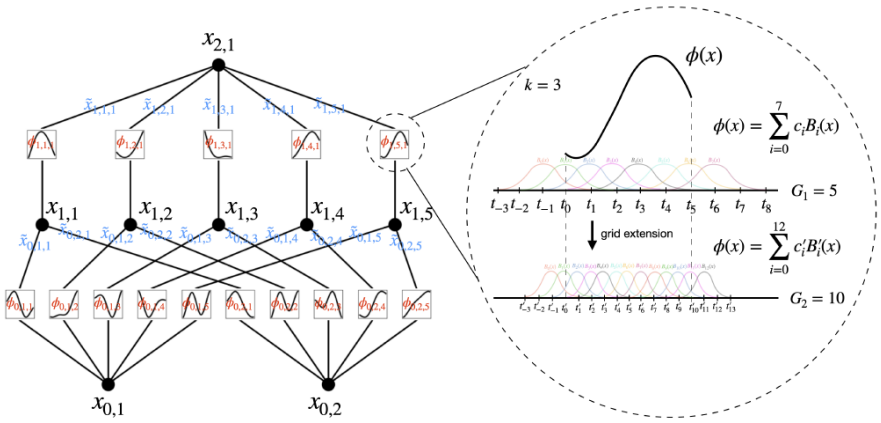

Figure 2.3 *KAN model architecture*

From an implementation perspective, the use of spline-based functions allows for efficient computation and gradient-based optimization. The model's parameters are the control points of these splines, which are differentiable and updatable through backpropagation.

On the other hand, the structure of MLPs can be expressed as:

$$\mathrm{MLP}(\mathrm{x}) = (\mathbf{W}_{L-1} \circ \sigma \circ \mathbf{W}_{L-2} \circ \sigma \circ \ldots \circ \mathbf{W}_1 \circ \sigma \circ \mathbf{W}_0)\mathrm{x} \tag{2.9}$$

Where each $\mathbf{W}_l$ is a weight matrix representing a linear transformation, and σ is a fixed nonlinear activation function.

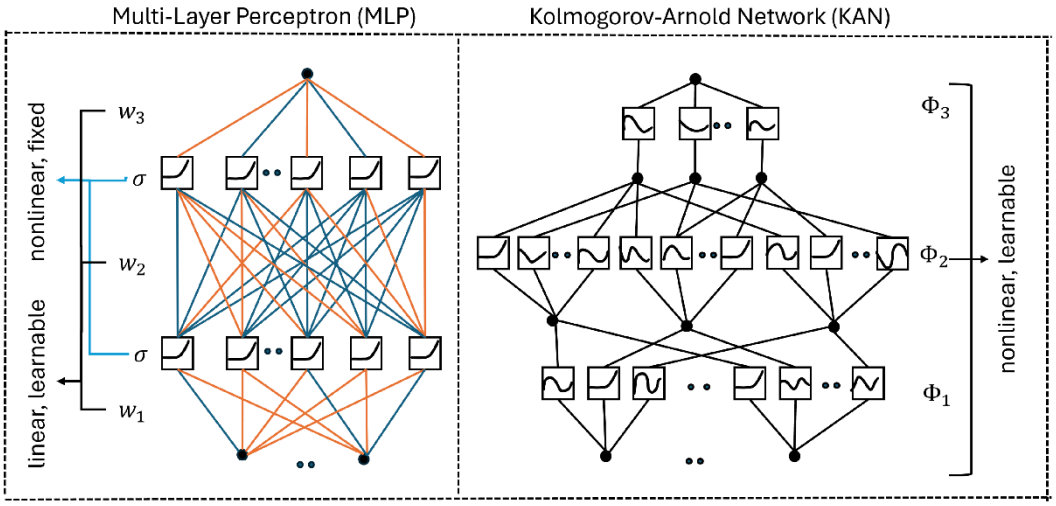

Figure 2.4 *The figure illustrates the structural comparison between MLPs and KANs.*

2.4 B-Splines in KAN

In KAN, the learnable univariate functions used along the edges of the network are often implemented using B-splines. A B-spline is a function made by smoothly joining several low-degree polynomial pieces, connected at specific points called knots. These knots divide the input range into intervals, within which the function acts like a polynomial. Across intervals, the transitions remain smooth and continuous. In this framework, each function $\phi_{q,p}$ and $\mathbf{\Phi}_q$ is expressed as a weighted sum of B-splines basis function. The general form for a spline in the context of KANs can be expressed using B-splines as follows:

$$\text{spline}(x) = \sum_i c_i B_i(x) \tag{2.10}$$

Where $B_i(x)$ are the spline basis functions and c_i are the trainable coefficients [7].

In B-spline interpolation, a univariate function is represented as a linear combination of basis functions defined over a sequence of knot points. The final output of a spline function at a given input value is obtained by evaluating the relevant basis functions and summing their weighted outputs. The weights, or control point coefficients, are trainable during training, these coefficients are updated to minimize the loss, while the knot locations typically remain fixed. This localized interpolation strategy enables smooth, flexible, and differentiable function learning, which aligns naturally with gradient-based optimization.

This can be visually illustrated in Figure 2.5.

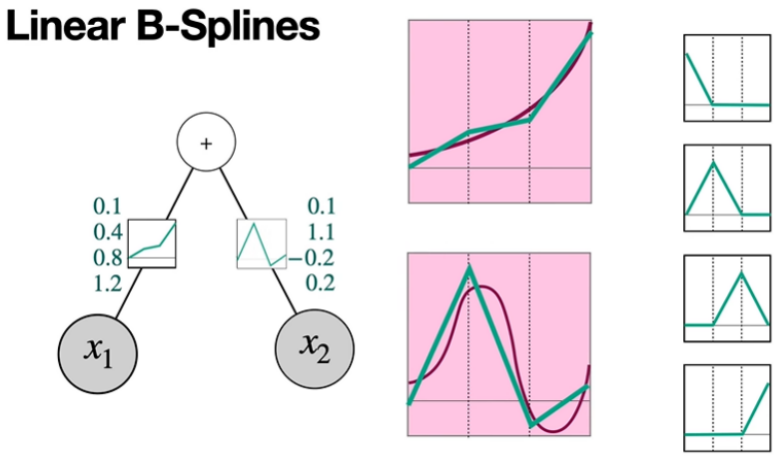

Figure 2.5 *B-splines in a function of KAN model.*

The number and the placement of knots determine the flexibility of the spline: more knots allow the function to capture more complex patterns, while fewer knots result in smoother curves. Since B-splines are differentiable, they can be optimized using gradient descent, which allows KAN to learn these functions directly from the data.

2.4.1 Comparison with Traditional Activation Functions

Unlike traditional activation functions such as the ReLU or sigmoid, which have fixed mathematical forms, B-splines are flexible and fully learnable. A ReLU function (formula 2.6), for example, does not change during training, and is used across all nodes in one

layer. In contrast, a B-spline function in KAN can adapt its shape during training by updating its weights c_i. This adaptability gives KAN an advantage in representing complex, nonlinear relationships without requiring deep architectures or a large number of parameters. Furthermore, B-splines provide local control, meaning that changes in one part of the input domain do not significantly affect other parts of the function. This may improve the stability and interpretability of the model.

In summary, splines are suitable for modelling low-dimensional functions because of their capacity for local adaption. In contrast, MLPs are more effective at capturing compositional and hierarchical patterns, but they often encounter difficulties when optimizing univariate functions.

2.5 Training of KAN

Training a KAN involves optimizing the parameters of its univariate functions to minimize a specific loss function. Unlike traditional neural networks, where weights in matrix multiplication are updated, KAN learns by adjusting the coefficients of spline basis functions along each edge of the network.

Each spline-based function in KAN, has a set of trainable coefficients c_i that determine the shape of the transformation. These coefficients are initialized randomly or with smooth priors and are updated using gradient-based optimization techniques such as stochastic gradient descent (SGD) or Adam. The model is trained end-to-end via backpropagation, leveraging the fact that B-splines are differentiable with respect to their coefficients.

During a forward pass, the input is transformed by univariate functions along the edges, aggregated at the nodes through summation, and passed layer by layer through the network. The final output is compared with the ground truth using a suitable loss function, such as mean squared error for regression [8] or cross entropy for classification [9]. In the backward pass, gradients of the loss are propagated through each spline function using the chain rule. This allows the model to adjust the coefficients to reduce the loss overtime. Although the internal structure differs, this training process is essentially similar to that of MLPs, which also rely on forward propagation, backpropagation, and gradient-based optimization.

To better understand how KAN operate in practice, let us consider an example of symbolic regression illustrated in Figure 2.6.

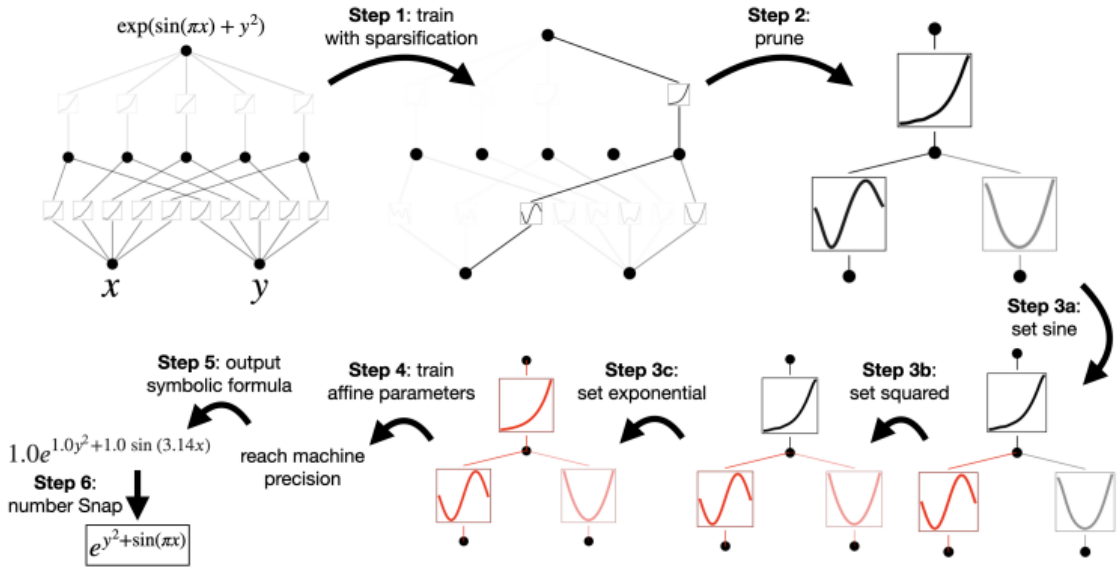

Figure 2.6 *An example of training KAN model.*

In this example, KAN is trained to recover the function $\exp(\sin(\pi x)+y^2)$ from the data. Initially, KAN is trained with a sparsity constraint [10]. This means that although the full network is dense, many of the univariate functions along the edges will learn to output near-zero values. These weak connections can later be pruned. After training, the insignificant connections are pruned from the network. This reduces complexity and focuses only on the most essential functional paths. At this point, the graph becomes smaller and easier to interpret. The remaining active functions are then examined and assigned symbolic identities based on their learned shape. In this example, the model first identifies a function behaving like $\sin(x)$, then another function approximate x^2, which corresponds to the y^2, and the outermost function is approximating $\exp(z)$, capturing the final nonlinear transformation. Once the symbolic structure is locked in, the affine parameters of each transformation are fine-tuned to match the original data more precisely. After optimizing the parameters, the KAN can now output a symbolic expression that closely matches the ground truth function.

KAN training tends to converge efficiently, especially in low-data regimes, due to its high expressiveness with relatively few parameters. Moreover, the local support of B-splines ensures that updates to the spline coefficients only affect nearby input regions, contributing to stable training and improved generalization. However, choosing the number of placement of knots should be carefully consider, as overly flexible splines may lead to overfitting [12], while too few may underfit the data [13].

2.6 Continual Learning

Continual learning, also known as lifelong learning, refers to the ability of a model to sequentially learn multiple tasks without forgetting previous acquired knowledge. This capability is particularly important in dynamic real-world environments where data arrives incrementally or evolves over time. A critical challenge in continual learning is catastrophic forgetting [14], where neural networks overwrite old information when trained on new data. KANs potentially mitigate this problem due to their local plasticity, derived from the local nature of spline. Specifically, since spline have localized support, each training sample influences only nearby spline coefficients, preserving far-away learned information [4]. By contrast, traditional neural networks like MLPs use global activation functions, causing even minor parameter updates to propagate broadly through the network net disrupts previously learned representations. This fundamental difference highlights a promising advantage of KANs for continual learning, allowing them to efficiently integrate new knowledge while maintaining the stability of earlier learned information.

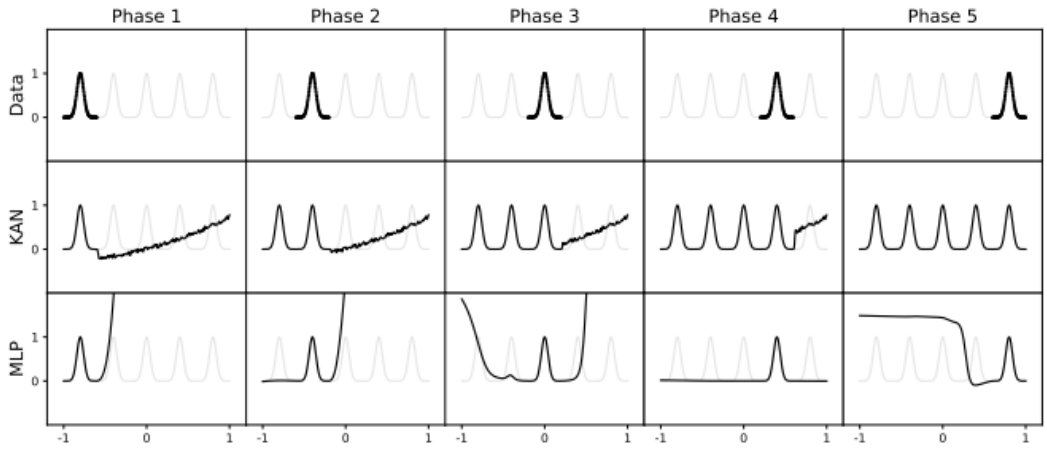

Figure 2.7 *Difference in continual learning between a KAN model and a MLP.*

3. IMPLEMENTATION

In this section, we detail the practical implementation of the proposed Kolmogorov-Arnold Network model, comparing it with traditional Multi-Layer Perceptron (MLP) architectures. The implementation process includes data preparation, model architecture design, training procedures, and evaluation protocols. Using PyTorch [16] as the primary framework, both KAN and MLP models were constructed and trained on benchmark datasets to assess their respective performances in function approximation tasks. All experiments conducted under consistent settings to ensure a fair and reliable comparison.

3.1 KAN Implementation (PyKAN)

In this experiment, we focus on evaluating the performance and behaviour of the KAN model before comparing it with the traditional MLP. To do so, we employ a synthetic regression task using a two-dimensional function defined as:

$$f(x_0, x_1) = \exp({x_0}^2 + {x_1}^2)$$

This function was chosen because it is smooth, continuous, and visually interpretable, making it well-suited for analysing the convergence behaviour of the model. The KAN model is configured with a width of $[2, 4, 1]$, which corresponds to two input features, a single hidden layer with four neurons, and one output (shown in figure 3.1). We also generate 1000 training samples randomly.

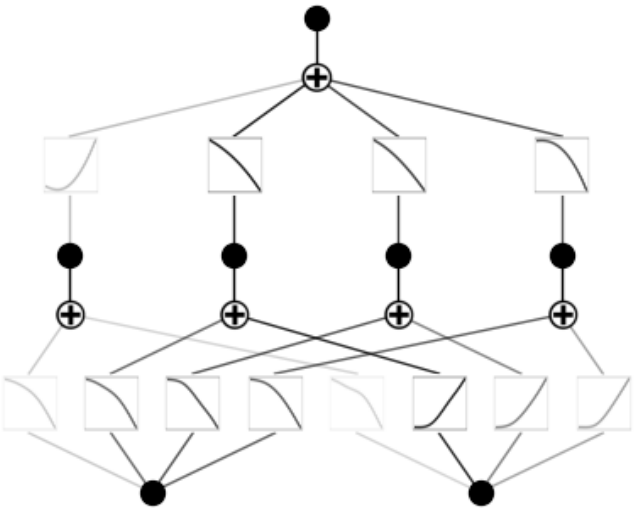

Figure 3.1 *KAN model with mentioned configuration before training.*

We use cubic splines ($k = 3$) and set the grid intervals per neuron to 3 ($\text{grid} = 5$). To further analyze the learning behaviors of the KAN model, we conducted a convergence experiment by varying the number of training samples and tracking both the training and test losses. Specifically, we compared two optimization methods: Adam and L-BFGS [18]. For each dataset size (ranging from 100 to 2000 samples), we trained the KAN model and evaluated its performance on both training test and a fixed, unseen test set. The results revealed a clear difference. While Adam optimizer consistently achieved lower training losses, its test losses were significant higher, especially on smaller datasets. This indicates a tendency to overfit the training data, as Adam is an adaptive first-order method that minimizes loss without necessarily encouraging smooth or globally consistent function shapes. In contrast, the L-BFGS optimizer, which approximates second-order curvature information, achieved slightly higher training losses but much more stable and lower test losses across all dataset sizes [19]. These findings show a critical trade-off between optimization speed and generalization quality, and they reinforce the suitability of L-BFGS for spline-based models like KAN.

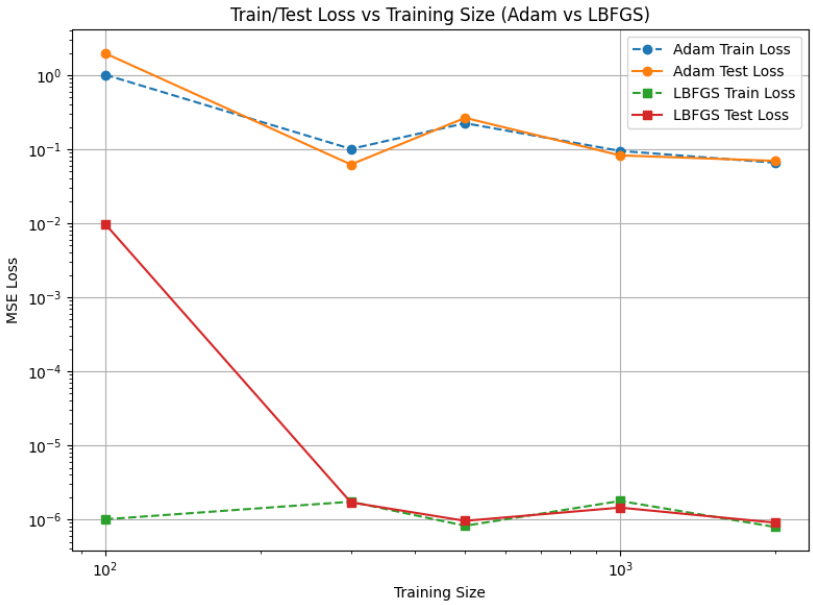

Figure 3.2 *MSE on the training and test sets for KAN model trained with Adam and L-BFGS optimizers.*

As a result, the structure learned by the KAN model using the L-BFGS optimizer is smooth and interpretable, as illustrated in the figure, clearly capturing the compositional of the target function.

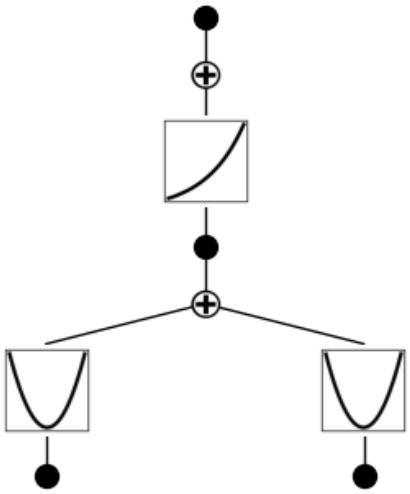

Figure 3.3 *The learned structure using L-BFGS is visually smooth and well-aligned with the target function.*

3.2 Dataset and Tasks

For evaluating and comparing the performance of KANs and MLPs, we selected the IMDb sentiment analysis dataset, a widely used benchmark in the field of natural language processing (NLP). The IMDb dataset consists of 50,000 movie reviews labelled as either positive or negative, evenly divided into training and testing subsets. This dataset was chosen because it presents realistic complexity, with varied review lengths and linguistic features that challenge the models to capture patterns in textual data. Using this dataset enables a clear comparison of both models' ability and it is aligning well with the theoretical and practical aims of this thesis. However, due to computational constraints and memory limitations, we reduced the dataset to 20000 training samples and 5000 testing samples (80% training and 20% test data).

3.3 Data Preprocessing

IMDb reviews were first cleaned and tokenized using standard preprocessing pipeline, including HTML tag and punctuation removal, lowercasing, and splitting into tokens. Word embeddings were learned using Word2Vec [15] with a vector size of 100, trained on the tokenized training corpus using a window size of 5 and a minimum frequency threshold of 2. Each review was represented as the average of its constituent word embeddings, resulting a 100-dimensional input vector.

After preprocessing, review vectors were converted to PyTorch tensors, and the corresponding binary sentiment labels (positive or negative) were also encoded to tensors

and stored as scalar targets. These tensors were then used as input for both the MLP and KAN models.

3.4 Model Architectures and Implementation

This section outlines the architectural design and implementation of KAN and MLP models used to evaluate performance on sentiment classification using the IMDb dataset. For the KAN model, we specifically leveraged PyKAN [17], a specialized Python library designed to simplify the creation and training of KAN architectures using spline-based transformations. PyKAN enables straightforward implementation of KAN by handling spline initialization, function fitting, and gradient based optimization efficiently. The loss function for both models were mean squared error (MSE). In the architectural notation used throughout this section, a configuration such as [100, 32 or 5 (for KAN), 1] represents a model with an input layer of size 100 (leftmost), a hidden layer (middle) with 32 neurons for MLP and for KAN it is the number of spline-based unit, and an output layer size of 1 (rightmost).

3.4.1 MLP Architecture

The Multi-Layer Perceptron model was implemented using fully connected layers with ReLU activation functions between hidden layers. To systematically evaluate the effect of model width and depth, we test four different MLP configurations: [100, 16, 1], [100, 32, 1], [100, 64, 1] and [100, 64, 32, 1]. These configurations were selected to represent increasing complexity in a controlled manner. The first three are shallow models with single hidden layer of increasing width (16, 32, and 64 units), allowing analysis of how widening the network affects performance and parameter usage. The final configuration adds a second hidden layer, providing insight into whether depth contributes significantly to performance on this task. Moreover, all MLP models were trained using the Adam optimizer with a learning rate of 1×10^{-3} for 30 epochs.

3.4.2 KAN Architecture

The Kolmogorov-Arnold Network replaces matrix multiplications with edge-wise spline transformations. Each connection is modelled as a univariate cubic spline, with fixed knot positions and trainable control point weights. The input passes through these spline functions individually, and each output unit computes a sum over transformed inputs, consistent with the Kolmogorov-Arnold representation. We evaluated three configurations of

KAN: [100, 4, 1], [100, 8, 1] and [100, 16, 1]. These configurations match the MLP experiments in structural simplicity and scalability. They were chosen to provide a fair comparison of parameter efficiency and performance across different widths. Unlike MLPs, where increasing width leads to more weight parameters and dense matrix operations, KAN scales in terms of the number of spline functions. These models used cubic spline with 5 grid points per input dimension. Training was performed using the L-BFGS optimizer over 30 steps, with a regularization term $\lambda = 0.0001$ and entropy penalty $\lambda_{\text{entropy}} = 0.5$ to control function complexity.

3.4.3 Evaluation and Metrics

After training, all models were evaluated on the test set using MSE. The number of trainable parameters was also recorded. Results were plotted with test MSE against model size (in log scale).

3.5 Results and Analysis

Figure 3.4 presents a comparison between KANs and MLPs in terms of terms of test mean squared error and the number of trainable parameters, plotted on logarithmic scales. Each point on the graph represents a different model configuration.

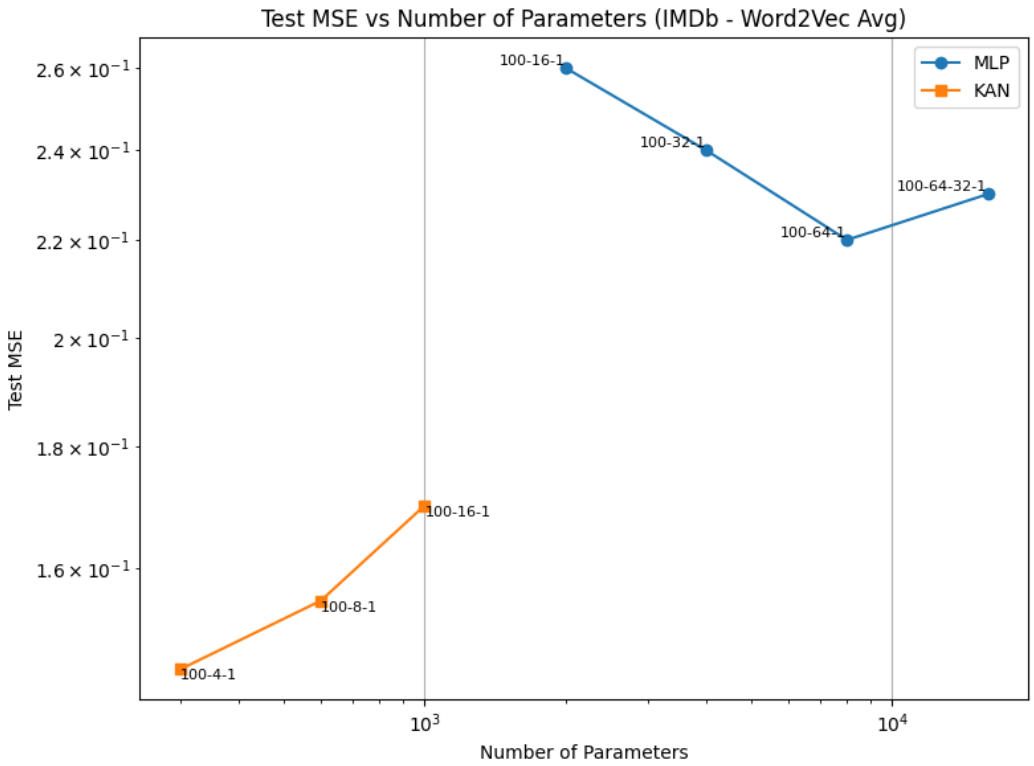

Figure 3.4 *Comparison of MLPs and KANs on the IMDb dataset.*

The comparison reveals that KAN models consistently achieve lower test MSE than MLPs, despite using significant fewer trainable parameters. As the MLP architecture is expanded with additional neurons or hidden layers, its test performance does improve incrementally, but these improvements are modest and come at the cost of rapidly increasing parameter count. In contrast, KAN models demonstrate a predictable and efficient scaling of performance, maintaining lower error rate across all tested configurations while preserving a compact model structure.

As the number of hidden units increase, MLP performance improves slightly but plateaus, while KAN performance begins to degrade slightly beyond 16 hidden units. This may reflect overfitting due to increased spline flexibility or increased sensitivity to regularization. Nevertheless, even the largest MLP ([100, 64, 32, 1]) does not outperform the mid-size KAN ([100, 8, 1]), reinforcing the idea that KANs can represent complex functions more compactly and with improved generalization.

The difference in parameter efficiency supports the theoretical strength of KANs, which are designed to approximate functions through localized, trainable univariate splines instead of global matrix-based computations. The test results confirm that KANs are highly effective at learning compact representations, and they are especially well-suited for tasks requiring smooth generalization or interpretability.

However, a significant drawback of the KAN architecture lies in its computational cost during training. While MLPs benefit from the simplicity and speed of sense matrix operations and first-order optimizer such as Adam, KANs involve more computationally intensive procedures. Each connection in KAN corresponds to a univariate spline transformation that must be evaluated and updated across a grid of control points, making both the forward and backward passes more complex. Moreover, the KAN models in this experiment were optimized using the L-BFGS algorithm, a second-order method that approximates curvature information and is inherently more resource-intensive than first-order optimizers.

This difference in computational demand becomes especially noticeable in environments without GPU acceleration. All experiments in this study were conducted on a CPU-only setup, which further amplifies the disparity in training time between two models. While the MLPs were able to complete training relatively quickly even across multiple configurations, the KAN models required substantially more time per training step. The combination of spline evaluation, entropy regularization, and second-order optimization made KANs significantly slower to train on CPU hardware.

Despite this limitation, the trade-off in training time may be justified in tasks where parameter efficiency, interpretability, and smooth generalization are prioritized over speed. The slower training performance of KANs can be understood as the computational price of learning structured, functional transformations rather than dense matrix operations.

4. CONCLUSIONS

This thesis presented a comparative study between Kolmogorov-Arnold Networks (KANs) and Multi-Layer Perceptrons (MLPs), with the aim of understanding their respective strengths and limitations in terms of function approximation, generalization, interpretability, and parameter efficiency. Grounded in the Kolmogorov-Arnold representation theorem, KANs provide an alternative neural architecture in which each edge is associated with a learnable univariate function, typically parameterized using splines. This design contrasts with the matrix-based structure of MLPs and offers a more interpretable and modular approach to learning complex mappings.

To evaluate both models, we implemented them using PyTorch and conducted experiments on two tasks: symbolic regression and sentiment classification using the IMDb dataset. In the symbolic regression task, we demonstrated that KAN models, especially when trained with the L-BFGS optimizer, achieved smoother and more generalizable approximations of the target function compared to MLPs. In the sentiment classification task, we compared the models across multiple configurations, examining test error and parameter count. The results clearly showed that KANs could achieve lower or comparable test error using significantly fewer trainable parameters. This supports the hypothesis that KANs are more parameter-efficient and better suited for learning compact representations.

However, our study also highlighted important trade-offs. KANs, while offering strong generalization and interpretability, incur a substantially higher computational cost during training. Their reliance on spline evaluations and second-order optimization makes them slower to train, particularly in CPU-only environments. In contrast, MLPs train faster and scale better computationally, making them more practical for time-sensitive or large-scale tasks.

In summary, this thesis confirms that KANs represent a promising architectural alternative to traditional MLPs, especially in tasks where interpretability, functional smoothness, and compactness are important. While they are not yet a universal replacement for MLPs, their mathematical foundation and empirical performance suggest valuable directions for future research. These include extending KANs to handle high-dimensional structured data, exploring adaptive knot placement, and applying KANs to continual or symbolic learning settings.

REFERENCES

[1] Online-PDH (n.d.) *Guide to Machine Learning for Engineers*.

[2] Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A.N., Gouws, S., Jones, L., Kaiser, Ł., Kalchbrenner, N., Parmar, N., Sepassi, R. and Shazeer, N. (2018) 'Stochastic gradient descent', in *Tensor2Tensor for Neural Network Translation and Classification*, section 2.2. arXiv preprint arXiv:1609.04747. Available at: https://arxiv.org/pdf/1609.04747.

[3] Kingma, D.P. and Ba, J. Adam: A method for stochastic optimization. ICLR, 2015.

[4] Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T. Y., and Tegmark, M. (2025) 'KAN: Kolmogorov-Arnold Networks'. arXiv preprint arXiv:2404.19756. Available at: https://arxiv.org/pdf/2404.19756.

[5] Mukherjee, S. (2024) 'Kolmogorov-Arnold Networks (KAN) Promising Alternative to Multi-Layer Perceptron?', *DigitalOcean Community Tutorials*, 17 September. Available at: https://www.digitalocean.com/community/tutorials/kolmogorov-arnold-networks-kan-revolutionizing-deep-learning.

[6] Danielso, K. (n.d.) '13.1 Multi-layer perceptrons (MLPs)'. Available at: https://kenndanielso.github.io/mlrefined/blog_posts/13_Multilayer_perceptrons/13_1_Multi_layer_perceptrons.html.

[7] Sheikh, H. (2024) 'Understanding Kolmogorov–Arnold Networks (KAN)', *TDS Archive*, 7 May.

[8] Encord (n.d) 'Mean Square Error (MSE)'. Available at: https://encord.com/glossary/mean-square-error-mse/.

[9] Hughes, C. (2024) 'A Brief Overview of Cross Entropy Loss', *Medium*, 25 September. Available at: https://medium.com/@chris.p.hughes10/a-brief-overview-of-cross-entropy-loss-523aa56b75d5.

[10] Dolgikh, S. (2021) 'Sparsity Constraint in Unsupervised Concept Learning', *CEUR Workshop Proceedings*, Vol. 2962, pp. 22–30. Available at: https://ceur-ws.org/Vol-2962/paper03.pdf (Accessed: 1 April 2025).

[11] Rakesh (2023) 'How Artificial Neural Networks Work: From Perceptrons to Gradient Descent', *Medium*, 5 August. Available at: https://medium.com/@rakeshandugala/how-artificial-neural-networks-work-from-perceptrons-to-gradient-descent-28c5552d5426.

[12] IBM (2021) 'What is overfitting?', *IBM Think Blog*, 15 October.

[13] IBM (2021) 'What is underfitting?', *IBM Think Blog*, 15 October.

[14] Nabil, M. (2023) 'Understanding and Mitigating Catastrophic Forgetting in Machine Learning', *Medium*, 8 October. Available at: https://medium.com/@navarai/understanding-and-mitigating-catastrophic-forgetting-in-machine-learning-d5caa93d375e.

[15] TensorFlow (n.d.) 'word2vec'. Available at: https://www.tensorflow.org/text/tutorials/word2vec.

[16] PyTorch (n.d.) *PyTorch*. Available at: https://pytorch.org/.

[17] Liu, Z. (n.d.) 'pykan: Kolmogorov Arnold Networks'. Available at: https://github.com/KindXiaoming/pykan.

[18] PyTorch (n.d.) 'torch.optim.LBFGS', *PyTorch Documentation*. Available at: https://pytorch.org/docs/stable/generated/torch.optim.LBFGS.html.

[19] Soham (2021) 'Linear regression with PyTorch: LBFGS vs Adam', *Medium*, 9 February. Available at: https://dragonbornmonk.medium.com/linear-regression-with-pytorch-lbfgs-vs-adam-19b62ce83d4