universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Using a Kolmogorov-Arnold Neural Network to distinguish Felsenstein Zone from Farris Zone

verfasst von | submitted by

Lea Sophie Scherer B.Sc.

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 875
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Bioinformatik
Betreut von | Supervisor:	Dr. Arndt von Haeseler

Abstract

This thesis investigates the application of a Kolmogorov-Arnold neural network (KAN), as proposed by Liu et al. [2024], to a phylogenetic classification problem: distinguishing between four taxon DNA alignments evolved under Farris or Felsenstein trees. These tree types differ primarily in branch length distribution, leading to different reconstruction outcomes depending on whether maximum parsimony or maximum likelihood methods are used.

Building on the work of Leuchtenberger et al. [2020], where a classical feedforward neural network was used for this task, this thesis explores whether a KAN can achieve comparable predictive performance while offering greater interpretability. The same dataset as in Leuchtenberger et al. [2020] was used to ensure direct comparability. The KAN was able to solve this task with the same performance as the network introduced by Leuchtenberger et al. [2020]. These results demonstrate that KANs are capable of solving the classification task effectively.

Zusammenfassung

Diese Arbeit untersucht die Anwendung eines Kolmogorov-Arnold neuronalen Netzes (KAN), wie es in Liu et al. [2024] präsentiert wurde, auf ein phylogenetisches Klassifikationsproblem: die Unterscheidung zwischen vier Taxon DNA-Alignments, die sich unter Farris- oder Felsenstein-Bäumen entwickelt haben. Diese Baumtypen unterscheiden sich in erster Linie in der Verteilung der Astlängen, was dazu führt, dass je nach verwendetem Verfahren, Maximum Parsimony oder Maximum Likelihood, unterschiedliche Rekonstruktionsresultate erzielt werden. Aufbauend auf der Arbeit von Leuchtenberger et al. [2020], in der für diese Aufgabe ein klassisches feedforward neuronales Netz eingesetzt wurde, wird in dieser Arbeit untersucht, ob ein KAN eine vergleichbare Vorhersageleistung erreichen kann und dabei zugleich eine größere Interpretierbarkeit bietet. Um eine direkte Vergleichbarkeit sicherzustellen, wurde derselbe Datensatz wie in Leuchtenberger et al. [2020] verwendet. Das KAN war in der Lage, diese Aufgabe mit derselben Leistung zu lösen wie das von Leuchtenberger et al. [2020] eingeführte Netz. Die Ergebnisse zeigen, dass KANs in der Lage sind, die Klassifikationsaufgabe effektiv zu bewältigen.

Contents

Chapter 1

Introduction and Motivation

A novel approach to neural network design, the Kolmogorov–Arnold network, was proposed by Liu et al. [2024]. This architecture promises improved interpretability and more efficient training compared to standard artificial neural networks.

In this thesis, the KAN is applied to the problem of distinguishing between Farris-type and Felsenstein-type phylogenetic trees. This task was previously addressed using a conventional artificial neural network (ANN) by Leuchtenberger et al. [2020]. Since both models tackle the same classification problem, their performance and efficiency will be compared, particularly with respect to layer size and architectural complexity.

Before implementing and applying the KAN, it is essential to understand how it differs from ANNs. This comparison provides insight into the potential advantages and limitations of the KAN architecture. In Chapter 2, relevant background information necessary for understanding the design, training, and evaluation of the KAN is provided. Chapter 3 outlines the methodologies

and resources used, including details on the datasets, network architecture, implementation, as well as the training and testing procedures. In Chapter 4, the trained models are analyzed and their results interpreted. Chapter 5 presents a summary of the findings, discusses limitations, and provides an outlook for future work.

Chapter 2

Background

To fully understand the structure and capabilities of Kolmogorov-Arnold networks, it is essential to first introduce and define the fundamental concepts. This section provides the necessary background knowledge.

Section 2.1 introduces artificial neural networks. Section 2.2 presents the structure of Kolmogorov-Arnold networks along with the underlying mathematical principles. The differences between KANs and traditional ANNs are discussed in Section 2.3. Section 2.4 defines the key terminology required for the subsequent chapters. Finally, Section 2.5 outlines the phylogenetic foundations relevant to the application of KANs explored in this thesis, including a description of the specific phylogenetic problem addressed.

2.1 Introduction to neural networks

A neural network is a system of units, called neurons, that processes information. Artificial neural networks can be viewed as a simplified model of

the human brain. They mimic the structure and function but do not fully replicate the complexity of the biological brain.

2.1.1 Artificial neural networks

A neural network can be thought of as a graph composed of nodes and edges, where the nodes are organized in layers. The input layer receives the initial data, which is then passed through one or more hidden layers for processing. The final result is produced by the output layer, which reflects how the network has interpreted or transformed the input (Goodfellow et al. [2016]).

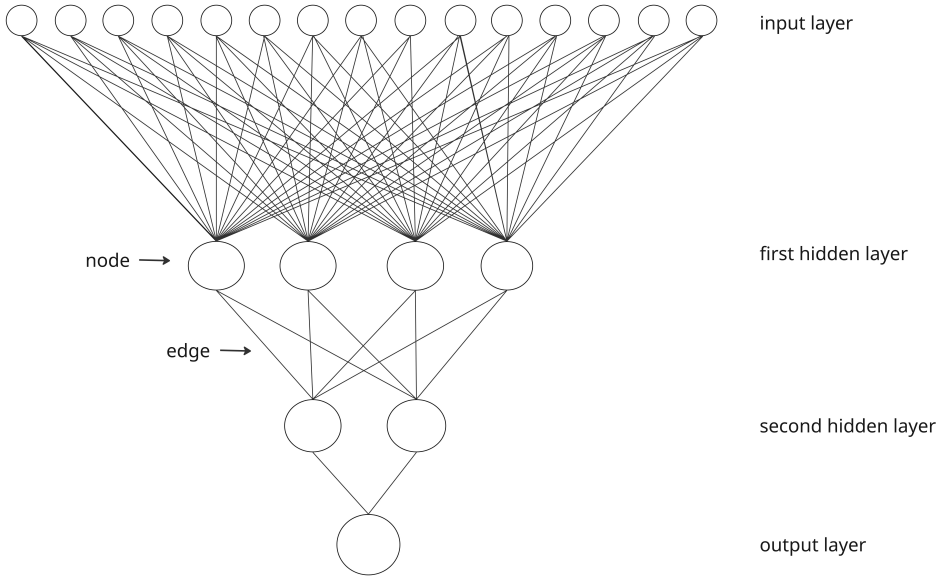

Figure 2.1: Simplified structure of an artificial neural network with two hidden layers.

Each node, also known as a neuron, receives numerical input, applies a weight, sums the results, and then passes the outcome through an activation

function. These activation functions introduce non-linearity, enabling the network to model complex relationships. The connections between neurons, called edges, give the weighted values from one layer to the next.

The network learns by adjusting these weights during a training process. Using the backpropagation algorithm (Rumelhart et al. [1986]), the network's output is compared to the expected result, the error is calculated, and then propagated backward through the network to update the weights. This iterative process gradually improves the network's accuracy.

2.1.2 Universal approximation theorem

The universal approximation theorems are foundational results in the theory of neural networks. Seminal work by Cybenko [1989], Hornik et al. [1989], and Pinkus [1999] established that feedforward neural networks are capable of approximating arbitrary continuous functions on compact subsets of $\mathbb{R}^n$, under fairly general conditions.

A representative version of the theorem for single-hidden-layer networks is the following (Hornik [1991]):

Theorem 2.1.1

Let $C(K)$ be the space of continuous functions on a compact set $K \subseteq \mathbb{R}^n$. Then, for every $f \in C(K)$ and every $\varepsilon > 0$, there exists a feedforward neural network g with a single hidden layer such that

$$|f(x) - g(x)| < \varepsilon \quad \textit{for all } x \in K. \tag{2.1}$$

This result implies that a feedforward neural network with a single hid-

den layer, a finite number of neurons, and a suitable activation function can approximate any continuous function arbitrarily well on a compact domain. This further suggests that even highly complex tasks, such as image recognition, can be solved using neural networks.

While Theorem 2.1.1 guarantees the existence of such a network, it does not specify how to construct it, how to choose the weights, or how many neurons are required to achieve a desired accuracy. In practice, approximating complex functions with a shallow network may require an impractically large number of neurons, leading to models that are computationally inefficient and difficult to train.

Therefore, modern deep learning often relies on architectures with multiple hidden layers, which can represent complex functions more compactly and effectively (Shrestha and Mahmood [2019]).

2.2 Kolmogorov-Arnold networks

A novel approach for neural networks was introduced by Liu et al. [2024]: the Kolmogorov-Arnold networks (KAN). The foundation for these networks is the Kolmogorov-Arnold representation theorem (Kolmogorov [1957]). Instead of basing the neural network on the universal approximation theorem, the Kolmogorov Arnold representation theorem is used.

2.2.1 The Kolmogorov-Arnold representation theorem

The Kolmogorov-Arnold representation theorem states that every multivariate continuous function can be represented as a finite composition of con-

tinuous univariate functions and addition. This implies that multivariate continuity does not inherently require multivariate structure for representation. Formally, this can be written as (Kolmogorov [1957]):

Theorem 2.2.1 (Kolmogorov-Arnold representation theorem) *Let $f : [0,1]^n \to \mathbb{R}$ be a continuous function. Then there exist continuous functions $\Phi_q : \mathbb{R} \to \mathbb{R}$, and continuous functions $\phi_{q,p} : [0,1] \to \mathbb{R}$, such that*

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right). \tag{2.2}$$

The architecture of a KAN follows directly from the structure suggested by the Kolmogorov–Arnold Representation Theorem. Denoting the input variables as $x_p, p = 1, ..., n$, the theorem shows that any continuous function can be exactly represented using univariate functions ϕ and Φ. Correspondingly, a KAN can be implemented with just two hidden layers. The first hidden layer consists of the function values $\phi_{q,p}(x_p)$ and the second hidden layer applies the outer functions Φ_q.

This construction implies that, in theory, any continuous function over a compact domain can be modeled exactly by a two-layer KAN. The bounds $2n + 1$ and n in Theorem 2.2.1 give upper limits on the number of functions and consequently, the number of edges theoretically required in such a representation. In practice, more functions than strictly necessary can be used, and these bounds serve primarily as theoretical guarantees rather than design limitations. It is also possible to use less functions, as one can just choose the remaining functions as zero.

To intuitively explain how to build the neural network based on the Kolmogorov-Arnold representation theorem consider the following example.

We would like to represent the following function

$$g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, x_1, x_2 \in \mathbb{R}, g(x_1, x_2) = x_1 \cdot x_2$$

with the help of the Kolmogorov-Arnold representation theorem. The theorem tells us, that there must be a representation

$$g(x_1, x_2) = \sum_{q=1}^{2\cdot 2+1} \Phi_q \left(\sum_{p=1}^{2} \phi_{q,p}(x_p) \right), \tag{2.3}$$

where $\Phi_q, \phi_{q,p}$ are continuous. By choosing the functions as

$$\begin{aligned}
&\phi_{1,1}(u) = u, && \phi_{1,2}(u) = u,\\
&\phi_{2,1}(u) = u, && \phi_{2,2}(u) = 0,\\
&\phi_{3,1}(u) = 0, && \phi_{3,2}(u) = u,\\
&\phi_{4,1}(u) = 0, && \phi_{4,2}(u) = 0,\\
&\phi_{5,1}(u) = 0, && \phi_{5,2}(u) = 0,\\
&\Phi_1(u) = \frac{1}{2}u^2, && \Phi_2(u) = -\frac{1}{2}u^2,\\
&\Phi_3(u) = -\frac{1}{2}u^2, && \Phi_4(u) = 0,\\
&\Phi_5(u) = 0,
\end{aligned}$$

the function $g(x_1, x_2)$ can be represented as

$$\begin{aligned} g(x_1, x_2) &= \overbrace{\Phi_1(\phi_{1,1}(x_1) + \phi_{1,2}(x_2))}^{\frac{1}{2}(x_1+x_2)^2} + \overbrace{\Phi_2(\phi_{2,1}(x_1) + \phi_{2,2}(x_2))}^{-\frac{1}{2}(x_1)^2} \\ &\quad + \overbrace{\Phi_3(\phi_{3,1}(x_1) + \phi_{3,2}(x_2))}^{-\frac{1}{2}(x_2)^2} \\ &= \frac{1}{2}(x_1 + x_2)^2 - \frac{1}{2}x_1^2 - \frac{1}{2}x_2^2 \\ &= x_1 \cdot x_2. \end{aligned} \tag{2.4}$$

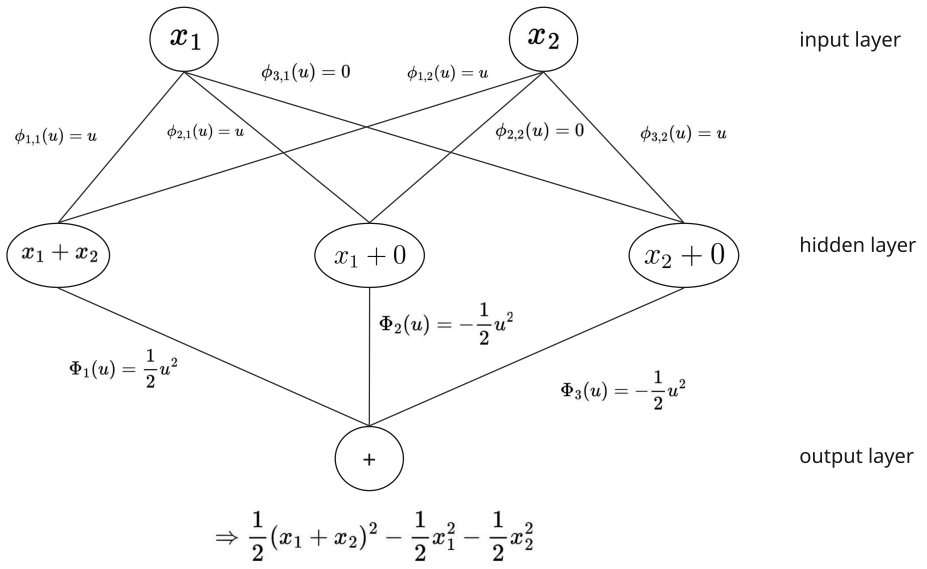

Figure 2.2: Visualized network architecture of the KAN example. For simplicity, the zero functions $\phi_{4,1}, \phi_{5,1}, \phi_{4,2}, \phi_{5,2}, \Phi_4, \Phi_5$ are omitted from this figure. For a more complex example see Liu et al. [2024].

The Kolmogorov–Arnold representation theorem is stronger than the Uni-

versal approximation theorem. While the latter ensures that a sufficiently large neural network can approximate any continuous function to arbitrary precision, the former asserts that an exact decomposition exists. Nevertheless, the Kolmogorov-Arnold representation theorem is non-constructive: it does not specify the actual form of the functions ϕ and Φ, which must be found or approximated in practice.

Identifying such functions is only practical in relatively simple cases, such as the one just shown. As the complexity of the network increases, deriving these functions analytically becomes intractable. To address this challenge, Liu et al. [2024] proposed using B-splines to approximate these functions. A detailed explanation of this approach is provided in Chapter 3.

2.3 Difference between KANs and ANNs

The most important difference between ANNs and KANs is the architecture of the networks. In ANNs the weights, sitting on the edges, are trained while in KANs the Φ and ϕ functions are trained.

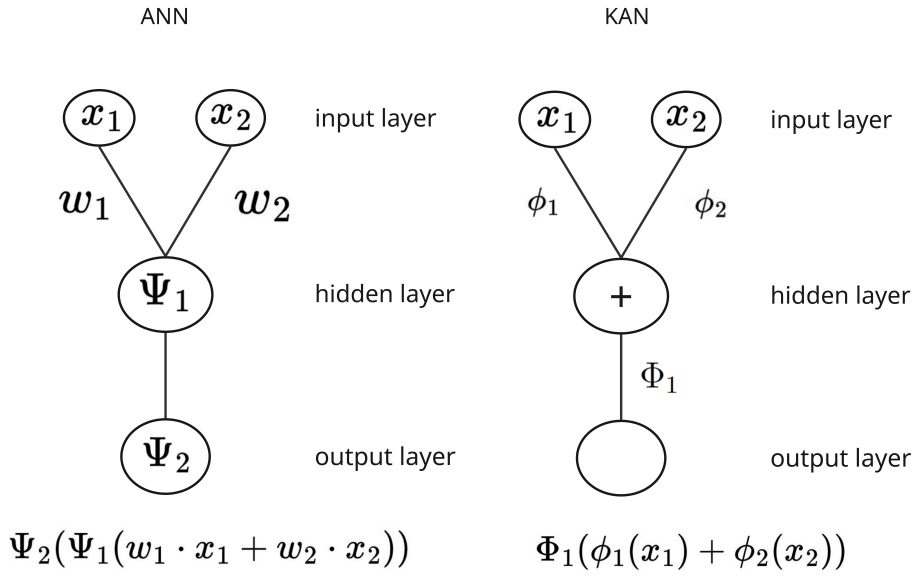

Figure 2.3: Comparison of the architecture of a simplified visualization of an artificial neural network and a Kolmogorov-Arnold network. Here the changeable component of the ANN would be the weights w_1 and w_2, operating on the edges, connecting the nodes. For the KAN, the functions ϕ_1, ϕ_2 and Φ_1 are trainable. In the ANN the activation functions, denotated as Ψ_1 and Ψ_2, are chosen beforehand and are fixed during training. The activation functions operate on the nodes.

2.4 Essential terminology in neural network theory

The following terms are important for understanding the training, testing and evaluation processes described in the upcoming chapters.

During the *training*, the neural network learns from labeled data by iteratively adjusting its parameters to minimize loss. The *loss* is a measure of the model's performance. It is computed using a loss function, which takes as input the model's predicted outputs and the true values, and returns a scalar value representing the prediction error (Goodfellow et al. [2016]).

Backpropagation (Rumelhart et al. [1986]) is an algorithm used to compute the gradient of the loss function with respect to each weight in the network. It applies the chain rule of calculus to propagate the error backwards through the network layers, making it possible to determine how each parameter contributed to the final loss. A more detailed explanation is given by Nielsen [2015]. *Gradient descent* is an optimization method that uses gradients to update the parameters in order to minimize the loss function. So backpropagation computes the necessary gradients, and gradient descent uses those gradients to iteratively update the model parameters and improve performance. The *learning rate* controls how much the neural network's parameters are adjusted at each optimization step. A high learning rate means that the model learns faster, but it may lead to instability or overshooting the optimal solution. A low learning rate results in slower learning but can lead to more precise convergence (Bengio [2012]).

During *validation*, the neural network is evaluated on a separate set of

labeled data that it has not seen during training, providing an unbiased estimate of its performance. *Overfitting* and *underfitting* describe how well a model learns the patterns during training. *Overfitting* occurs when there is a large difference between training error and testing error. This can happen when the model learns the training data too well, including outliers and noise, which can result in good performance during training but poor performance on unseen data. *Underfitting* describes the model having a large error on the training data (Goodfellow et al. [2016]), meaning the model is not able to learn the underlying relationships from the data. *Testing* refers to applying the trained network to new input data to obtain outputs such as class labels or numerical estimates (Goodfellow et al. [2016]).

2.5 Basics in phylogenetics

This section introduces background knowledge on phylogenetics, which is necessary for understanding the subsequent application of the KAN.

2.5.1 Essential terminology in phylogenetics

The study of evolutionary relationships between species, genes, or other biological entities is known as phylogenetics. By examining heritable characteristics like DNA, it aims to recreate the branching patterns of evolution (Haber and Velasco [2024], Lohrer [2022]).

In several branches of biology, including genomics and evolutionary biology, an understanding of phylogenetic relationships is essential. It enables scientists to examine evolution, determine species classifications, track the

origins of features, and even predict gene functions.

Phylogenetics uses a combination of biological data and mathematical modeling to examine such relationships. The trees that best explain the observed similarities and differences in genomic sequences are inferred using computational techniques. This procedure, called phylogenetic inference, requires a thorough comprehension of several fundamental ideas, which are described here. The following definitions can be looked up in even more detail in Christensen [2023].

A *phylogenetic tree* is a representation of the evolutionary relationships between different species or units (taxa) sharing a common ancestor. Each inner node represents a common ancestor of the descendants (leaf). Edge lengths often correspond to the estimated time of splitting or the number of mutations. *Phylogenetic inference* includes methods for determining degrees of relatedness between species or individuals of a species. The process begins with the collection of DNA, RNA, or protein sequences, which are aligned to identify homologous sites, ensuring positional correspondence across sequences. Following the *alignment*, an appropriate substitution model is selected to account for evolutionary processes. Phylogenetic trees are then reconstructed using algorithms such as maximum parsimony or maximum likelihood.

Inner branches represent connections between internal nodes that represent evolutionary relationships between common ancestors. *External branches* are endpoints of the tree that represent current species or sequences with no further descendants. *Nodes* are points in the tree where branches diverge. Inner nodes represent hypothetical common ancestors. *Leaves* (outer nodes)

represent observed species or sequences.

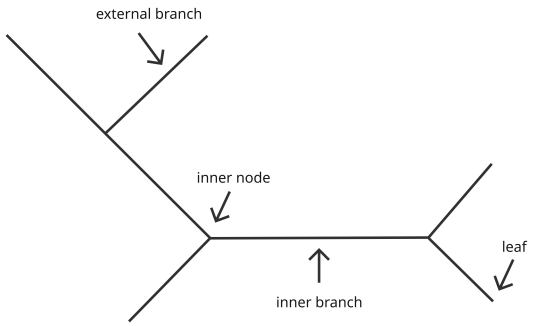

Figure 2.4: phylogenetic tree with notations.

2.5.2 Tree reconstruction methods

Phylogenetic tree reconstruction methods aim to infer the evolutionary relationships among species, genes, or other taxa based on molecular sequence data (De Bruyn et al. [2014]). Phylogenetic inference methods take an alignment of sequences as input, and produce a phylogenetic tree as output, representing a hypothesis of the evolutionary relationships among the sequences. Two commonly used methods are maximum parsimony (MP) and maximum likelihood (ML).

MP, initially proposed by Edwards and Cavalli-Sforza [1964], seeks the tree that requires the fewest evolutionary changes to explain the observed sequence data. It is based on the principle that the simplest explanation, so the one minimizing total nucleotide substitutions, is best. Algorithms such as those developed by Fitch [1971] and later by Sankoff [1975] allow for calculating this minimum efficiently.

On the other hand, ML introduced into phylogenetics by Felsenstein [1981], takes a statistical approach. It evaluates various tree topologies by calculating the probability (likelihood) that a particular tree would give rise to the observed sequence alignment, given a model of sequence evolution. The tree with the highest likelihood is then chosen as the best estimate.

In this thesis, the Jukes-Cantor model is used to model sequence evolution, which assumes equal base frequencies and uniform mutation rates across all nucleotide substitutions (Jukes and Cantor [1969]).

2.5.3 The Felsenstein zone and the Farris zone

The distinction between Felsenstein-type and Farris-type trees is essential when examining the reliability of phylogenetic inference methods. Both tree types are defined for four taxa and differ in how the long and short external branches are arranged. Each tree consists of an internal branch of length p, two external branches of length p, and two external branches of length q. In Farris-type trees, the two long branches are grouped together, whereas in Felsenstein-type trees, each long branch is paired with a short one (Leuchtenberger et al. [2020]).

When the evolutionary process follows a Felsenstein-type tree and the substitution probability along long branches greatly exceeds the substitution probability along the short ones, MP often misidentifies the tree topology. Instead of correctly pairing long with short branches, MP tends to group the two long branches together, effectively reconstructing a Farris-type tree. This misinterpretation is known as *long-branch attraction* (Felsenstein [1978]). The range of branch lengths where MP is prone to this error is

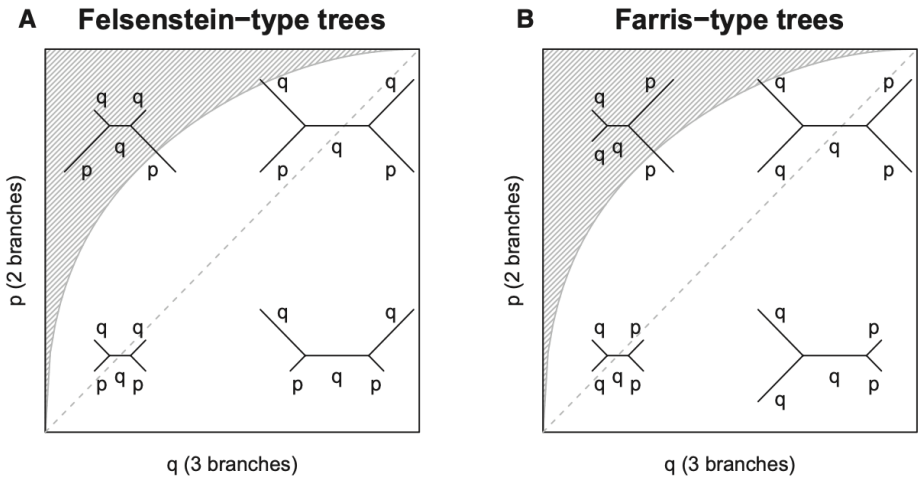

Figure 2.5: Felsenstein-type tree (left) and Farris-type tree (right) across varying branch-length parameters p and q. The shaded area denotes the Felsenstein zone. Source: Leuchtenberger et al. [2020].

referred to as the *Felsenstein zone*. Notably, this error persists even with arbitrarily long alignments, indicating that MP is statistically inconsistent in this region.

Moreover, it is important to distinguish between the true phylogenetic tree, which represents the actual evolutionary relationships and history of species, and the predicted phylogenetic tree, which is the inferred tree reconstructed from genetic data using computational methods. Long-branch attraction affects the predicted tree, causing it to incorrectly connect distantly related species with long branches, thereby deviating from the true evolutionary history (Felsenstein [1978]).

When data evolves under a Farris-type tree with certain branch length configurations, ML may favor an incorrect Felsenstein-type tree. In this case,

ML separates the two long branches and instead pairs each with a short one. This phenomenon is called *long-branch repulsion*, and the parameter space where it occurs is known as the *Farris zone* (Waddell [1995], Siddall [1998]). Unlike long-branch attraction, the impact of long-branch repulsion diminishes with increasing alignment length, and ML eventually converges to the correct topology. Thus, ML remains statistically consistent in the limit of infinite data (Yang [1994]).

In practical scenarios with real (and finite-length) sequence data, distinguishing whether an incorrect reconstruction is due to long-brach attraction or repulsion is nontrivial. For instance, if MP yields a Farris-type tree and ML returns a Felsenstein-type tree, it is unclear which method has inferred the correct tree. This uncertainty underscores the need for approaches that can identify whether an alignment originated from a Farris-type or Felsenstein-type tree.

Chapter 3

Resources and Methods

The aim of this thesis is to evaluate whether a KAN can distinguish between multiple sequence alignments evolved under Farris- and Felsenstein-type trees. In this chapter the steps for building, training and testing the KAN are presented. All codes, test and training data can be found on GitHub (`https://github.com/Cibiv/quartetKAN.git`).

Section 3.1 introduces the overall approach and outlines the KAN's primary objectives. Section 3.2 provides a detailed description of the structure and composition of the training and testing datasets. In Section 3.3, the architecture of the KAN is explained. Section 3.4 shows how the activation function for KANs are calculated, by discussing a specific example. Section 3.5 explains the technical implementation for building, training and testing of the KAN.

3.1 Approach

To perform the training and testing process of KANs large numbers of simulated sequence alignments are generated, on either a Felsenstein- or a Farris-type tree. These trees have branch lengths p, q between 0 and 0.75, in order to recreate scenarios that are characteristic of the two zones. The network is trained using frequency vectors as input, which represent the relative frequencies of different site-pattern categories derived from the alignments. When classifying previously unseen alignments, the network is provided with the corresponding frequency vectors and predicts whether the alignment is likely originated from a Felsenstein-type tree or a Farris-type tree. If successful, the KAN recognizes characteristic distribution patterns within the frequency data that are typical for each tree topology.

3.2 Datasets

The training and testing datasets are generated exactly as described by Leuchtenberger et al. [2020]. Alignment simulation followed the Jukes-Cantor model of molecular sequence evolution, which assumes uniform mutation rates across all nucleotides (Jukes and Cantor [1969]). As input for the network a frequency vector is used, which provides a numerical summary of sequence alignments based on the occurrence of fifteen site patterns. These patterns represent the different combinations of nucleotides found at the four positions corresponding to the taxa in an alignment. For DNA sequences from four taxa and possible nucleotides (A, C, G, T), there are 256 possible site patterns. Due to the symmetries in the substitution model, these 256

patterns can be grouped into 15 distinct categories.

Table 3.1: Pattern categories based on nucleotide similarity across four taxa ($x, y, w, z \in \{A, C, G, T\}$).

Explanation	**Categories**
All four taxa identical	xxxx
Three identical, one different	xxxy, xxyx, xyxx, yxxx
two identical pairs	xxyy, xyxy, xyyx
Two identical, two different	xxyz, xyxz, xyyz, xyzx, xyzy, xyzz
All different	xyzw

For a pair of branch parameters (p, q), one can calculate the probabilities associated with the different site patterns (Felsenstein [2004], p. 111). These probabilities define a multinomial distribution in 15 dimensions, denoted MD(p, q). The parameters p and q describe the chance that a substitution occurs along each respective branch, thereby determining the expected frequencies of the 15 pattern classes (Leuchtenberger et al. [2020]).

The frequency vector captures information about site pattern distributions that reflect the underlying phylogenetic tree. The neural network is trained to recognize these differences. Using simulated data labeled by tree type (Felsenstein or Farris), it learns which pattern distributions are characteristic of each. At prediction time, the frequency vector of a new alignment is computed and passed to the network, which classifies it based on learned distinctions between the two tree types.

The data generation used unrooted four-taxon trees, assigning two branches mutation rate p and the remaining three branches rate q (see Figure 2.5). For the training set, both p and q were varied between 0.005 and 0.745 in 0.01 increments, resulting in 5,625 branch length combinations. For each (p, q) com-

bination, 1,000 sequence alignments were simulated (Leuchtenberger et al. [2020]). For each alignment 1,000 MD(p, q) multinomially distributed site patterns were sampled (Leuchtenberger et al. [2020], Felsenstein [1978])).

To train the neural network without bias from the specific order of taxa, each generated alignment underwent 24 permutations, representing all possible arrangements of the four taxa (Leuchtenberger et al. [2020]). The alignments were transformed into input frequency vectors.

For the test data, p and q were varied over a coarser grid, from 0.025 to 0.725 in increments of 0.05, leading to 225 parameter combinations. For each combination, 200 multiple sequence alignments, each 1,000 nucleotides in length, were simulated using the program Seq-Gen (Rambaut and Grass [1997], Leuchtenberger et al. [2020]).

3.3 Network architecture and parameters

A neural network has two different classes of parameters: hyperparameters and model parameters. Hyperparameters are set before training begins and are not learned from the training data. Some examples are the number and size of hidden layers, the learning rate, batch size, number of epochs, and the loss function, which measures the discrepancy between the predicted and true output values. Within the framework of the training process, different combinations of hyperparameters were tested. The results are discussed in Chapter 4.

The network receives as input the 15-dimensional frequency vector. This input is propagated sequentially through a number of hidden layers, whose

number and sizes are defined by the hyperparameters. To provide a concrete visualization a simple network with two hidden layers and a manageable number of neurons was chosen, which will be consulted as an example in the upcoming chapters. Figure 3.1 shows the KAN with shape $[15, 4, 2, 1]$. The model will be referred to as KAN_[15,4,2,1]. To distinguish which layer ϕ is applied to, an additional index $\phi_{\ell,i,j}$, where ℓ denotes the layer, was added.

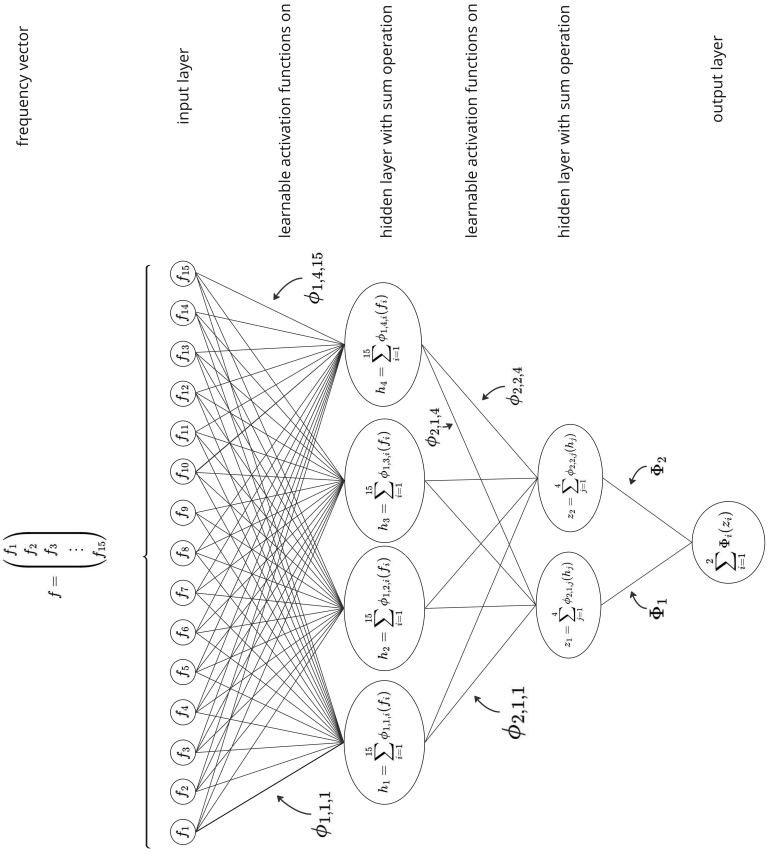

Figure 3.1: Visual representation of the structure for a KAN with layer structure [15,4,2,1]. Each edge corresponds to a learnable B-spline function $\phi_{\ell,i,j}$.

Since the Kolmogorov–Arnold representation theorem does not specify how the required univariate functions should be constructed, Liu et al. [2024] proposes to approximate them using B-spline-based functions. The learnable $\phi_{\ell,i,j}$ functions are implemented as linear combinations of B-splines.

B-splines are piecewise polynomial functions defined over a fixed grid, which divides the input domain into intervals. The number of intervals determines the resolution of the spline. A finer grid allows the network to represent more complex functions, while a coarser grid reduces the number of parameters and helps to avoid overfitting. B-splines act locally and can represent smooth univariate functions with high flexibility. Formally B-splines are defined as follows (de Boor [2001], de Boor [1972]).

Definition 3.3.1 (B-splines)
Let $t = (t_0, t_1, \ldots, t_{n+k+1})$ be a non-decreasing sequence of real numbers. The B-splines $B_{i,k}(x)$ of degree $k \geq 0$ are defined recursively:

Degree $k = 0$:

$$B_{i,0}(x) = \begin{cases} 1, & \text{if } t_i \leq x < t_{i+1},\ i = 0, ..., n+k \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

Degree $k \geq 1$:

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x), \tag{3.2}$$

Each function $B_{i,k}(x)$ is a piecewise polynomial of degree k with support on the interval $[t_i, t_{i+k+1})$. The B-splines are only dependent on the choice

of grid and have the following properties:

i **Local support**: $B_{i,k}(x) = 0$ for $x \notin [t_i, t_{i+k+1})$.

ii **Non-negativity**: $B_{i,k}(x) \geq 0$ for $x \in [t_i, ...t_{i+k+n})$.

iii **Partition of unity**:

$$\sum_{i=0}^{n} B_{i,k}(x) = 1 \quad \text{for all } x \in [t_k, t_{n+1}]. \tag{3.3}$$

Further each edge function $\phi_{\ell,i,j}$ in a KAN is defined as a weighted sum of B-splines $B_{r,k}(x)$ of order k (Liu et al. [2024]):

$$\phi_{i,j}(x) = \sum_{r} c_r \cdot B_{r,k}(x), \tag{3.4}$$

where c_r are trainable coefficients that are the model parameters of the network and r is the number of basis functions. These coefficients are randomly initialized at the start of training.

Figure 3.1 visualizes how in each hidden layer, the values from all input neurons are passed through the corresponding edge functions $\phi_{\ell,i,j}$. These transformed outputs are then summed at each neuron. Since the nonlinearity is entirely captured by the B-splines on the edges, classical node-based activation functions are no longer necessary.

For the specific phylogenetic application in this thesis, the outputs of the final hidden layer are passed through an activation function and summed to yield a single output value between 0 and 1. For this, the sigmoid function,

defined as (Mitchell [1997], page 97)

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{3.5}$$

which smoothly maps any real-valued input to the interval $(0, 1)$, was chosen. Here, the final output represents the network's confidence that the input corresponds to a Felsenstein- or a Farris-type tree. A value near 0 suggests a classification as Felsenstein, while a value near 1 suggests Farris.

3.4 How are the ϕ functions calculated in KANs?

The $\phi_{\cdot,i,j}$ functions are formed by calculating $\phi_{\cdot,i,j}(x) = \sum_r c_r B_{r,k}(x)$, where the $B_{r,k}$ are the B-splines. For KAN_[15,4,2,1] the grid was chosen as

$$[-2.2, -1.8, -1.4, -1, -0.6, -0.2, 0.2, 0.6, 1, 1.4, 1.8, 2.2]. \tag{3.6}$$

The resulting B-splines for $k = 1, 2, 3$ are shown in Appendix A. Note that the trainable and varying part of the ϕ functions operating on KANs edges are the c_r.

3.4.1 Detailed example of a function ϕ

To enhance clarity and provide a concrete illustration of the functions operating along the edges of the hidden layers, a specific example from the previously discussed KAN_[15,4,2,1], depicted in Figure 3.1 is used. In particular, the function $\phi_{1,1,1}$, which corresponds to the edge highlighted in Figure 3.2, is shown in detail in this section.

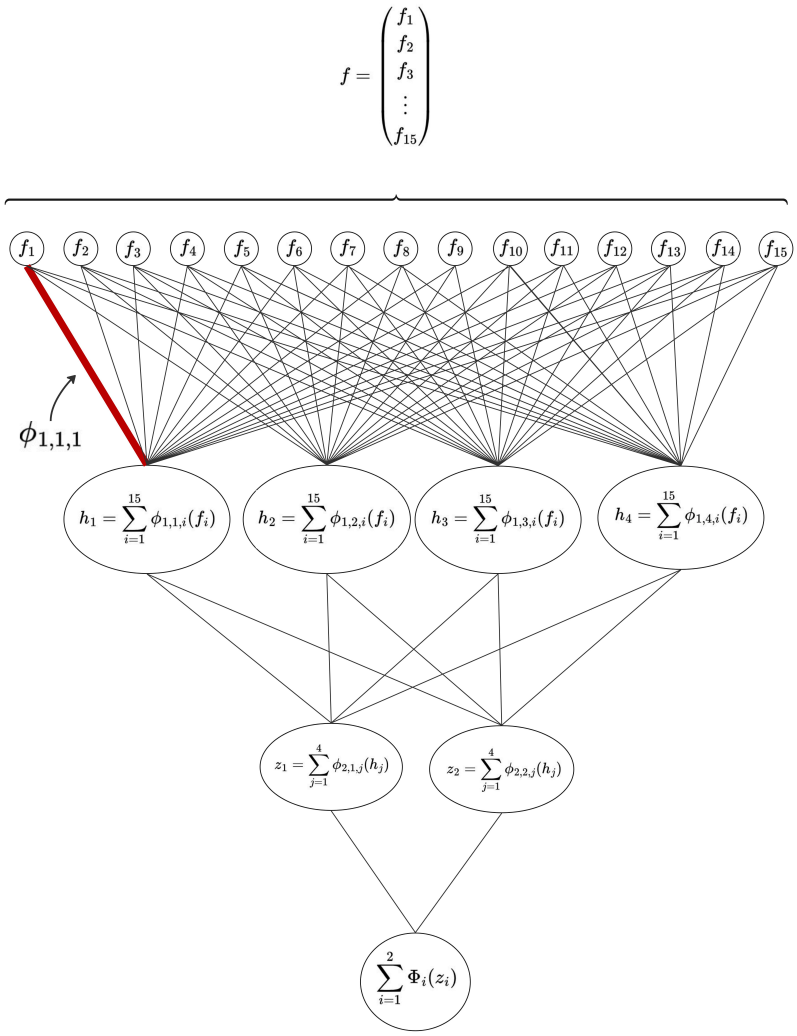

Figure 3.2: Visualization of the edge where the function $\phi_{1,1,1}$ is applied in KAN_[15,4,2,1].

The function $\phi_{1,1,1}(f_1)$ operates on the input feature f_1 from the input layer and provides its output to the first neuron h_1 in the first hidden layer.

Since cubic B-splines are used in this model (i.e., spline degree $k = 3$), the grid consists of 12 values and the number of basis functions is determined by the relation $k + r + 1 = 12$ (Piegl and Tiller [1997]), $\phi_{1,1,1}$ includes $r = 8$ basis functions. Thus, $\phi_{1,1,1}$ is expressed as

$$\phi_{1,1,1}(f_1) = \sum_{r=1}^{8} c_r \cdot B_{r,3}(f_1). \tag{3.7}$$

The code needed to extract c for a specific edge in a model is provided on GitHub (`https://github.com/Cibiv/quartetKAN.git`). Here, the coefficient vector $\mathbf{c} = \sum_{r=1}^{8} c_r$ is:

$$\mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{pmatrix} = \begin{pmatrix} 0.207415462 \\ 0.030397773 \\ 2.27792001 \\ 0.141335964 \\ 0.054345883 \\ -0.408105105 \\ -1.61672843 \\ 1.38897979 \end{pmatrix}. \tag{3.8}$$

Given an examplatory input frequency vector

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \\ f_{11} \\ f_{12} \\ f_{13} \\ f_{14} \\ f_{15} \end{bmatrix} = \begin{bmatrix} 0.079 \\ 0.019 \\ 0.135 \\ 0.019 \\ 0.031 \\ 0.040 \\ 0.129 \\ 0.040 \\ 0.020 \\ 0.142 \\ 0.041 \\ 0.032 \\ 0.190 \\ 0.038 \\ 0.045 \end{bmatrix} \tag{3.9}$$

where $f_1 = 0.079$ and substituting this into the equation for $\phi_{1,1,1}$ yields

$$\begin{aligned} \phi_{1,1,1}(0.079) = {} & 0.207415462 \cdot B_{0,3}(0.079) + 0.030397773 \cdot B_{1,3}(0.079) \\ & + 2.27792001 \cdot B_{2,3}(0.079) + 0.141335964 \cdot B_{3,3}(0.079) \\ & + 0.054345883 \cdot B_{4,3}(0.079) + (-0.408105105) \cdot B_{5,3}(0.079) \\ & + (-1.61672843) \cdot B_{6,3}(0.079) + 1.38897979 \cdot B_{7,3}(0.079). \end{aligned} \tag{3.10}$$

Based on the support of the B-spline basis functions (see Appendix A), most terms vanish at $f_1 = 0.079$, and only the following terms contribute as their support contains 0.079:

$$\begin{aligned}\phi_{1,1,1}(0.079) = {} & 0.141335964 \cdot B_{3,3}(0.079) \\ & + 0.054345883 \cdot B_{4,3}(0.079) \\ & - 0.408105105 \cdot B_{5,3}(0.079)\end{aligned} \tag{3.11}$$

The basis functions evaluate to

$$\begin{aligned}B_{3,3}(0.079) &= \frac{1}{48}(23 - 75 \cdot 0.079 - 75 \cdot 0.079^2 + 375 \cdot 0.079^3), \\ B_{4,3}(0.079) &= \frac{1}{48}(23 + 75 \cdot 0.079 - 75 \cdot 0.079^2 + 375 \cdot 0.079^3),\end{aligned} \tag{3.12}$$

and

$$B_{5,3}(0.079) = \frac{1}{48}(1 + 5 \cdot 0.079)^3. \tag{3.13}$$

And plugging in these values, the result is:

$$\begin{aligned}\phi_{1,1,1}(0.079) &\approx 0.14134 \cdot 0.3491 + 0.05435 \cdot 0.5967 - 0.40811 \cdot 0.0566 \\ &\approx 0.04944 + 0.03243 - 0.02308 \\ &= \mathbf{0.05879}.\end{aligned} \tag{3.14}$$

This procedure can be repeated for each of the 15 input features using their corresponding functions $\phi_{\ell,1,j}$, yielding 15 values whose sum is the output of neuron h_1 in the first hidden layer.

3.5 Implementation

To implement the KAN with Python, TensorFlow and Keras are used (Abadi et al. [2015], Chollet et al. [2015]). As hidden layers of the KAN, DenseKAN layers were used, which are KAN-style dense layers that replace each scalar linear weight by a learnable B-spline based function on the corresponding edge. These were implemented using a customized version of the `dense.py` module from the open-source `tfkan` package, available on GitHub (Zhou [2024]). The `tfkan` library is a TensorFlow-based implementation of the Kolmogorov–Arnold networks by Liu et al. [2024].

To integrate the DenseKAN layers into the KAN's architecture and ensure compatibility with the rest of the codebase and environment, the original `dense.py` file was slightly modified. These modifications were minimal and primarily aimed at adapting the class structure, interfaces, and dependencies to match the specific model setup and TensorFlow as well as Keras versions used in this thesis. All code can be found on GitHub (`https://github.com/Cibiv/quartetKAN.git`).

3.5.1 Training of KAN

The KAN was trained using standard backpropagation to minimize the loss function while maximizing classification accuracy. Datasets were generated as described in Section 3.2. 90% of the data was used for training and the remaining data was reserved for validation. Throughout the training, the KAN is evaluated on training and validation data to check if over- or underfitting is present. For evaluation, a binary classification metric was

used, with a default decision threshold of 0.5 to distinguish between the two classes: Felsenstein (< 0.5) and Farris (> 0.5).

3.5.2 Testing of KAN

To evaluate the performance of the trained KAN, a testing procedure is implemented. The test data consists of simulated alignments as described in Section 3.2. During the training and during the validation phase the classification accuracy is measured. Thereby the proportion of alignments for which the KAN inferred the correct tree type was determined. During the analysis the overall accuracy on the test alignments was measured as well as the accuracy only on those evolved under a Farris-type tree (or Felsenstein-type tree respectively). To facilitate visual interpretation, the accuracies for both trees and each (p, q) combination were plotted as two-dimensional heatmaps for all (p, q) combinations. These plots display the KAN's classification accuracy under various evolutionary conditions and provided a clear depiction of areas of strength and weakness.

Chapter 4

Results and Analysis

This chapter presents the results of the training and provides an analysis of the evaluated KANs. Section 4.1 introduces key terminology used to evaluate the models' performance. In Section 4.2, a specific model is selected for an in-depth analysis, including visualizations of loss and accuracy over training epochs, as well as the evaluation on the test data. Section 4.3 outlines the test results and examines the impact of various hyperparameter configurations. Section 4.4 presents a comparative evaluation of the KAN and ANN, examining differences in performance.

4.1 Definitions of accuracy metrics

Accuracy Farris refers to the accuracy of a network when applied to alignments that have evolved under Farris-type trees. *Accuracy Felsenstein* is analogous to *accuracy Farris*, but applies to alignments that evolved under Felsenstein-type trees. *Accuracy average* denotes the overall accuracy of a network across all alignments, regardless of whether they evolved under Farris-type or Felsenstein-type trees. It provides a general assessment of the networks's performance.

4.2 Analysis and interpretation of a specific model

To demonstrate the effectiveness of the training and evaluation process, we examine KAN_[15,4,2,1], previously introduced in Section 3.4.1. The learning rate was chosen as 0.0001 and the batch size as 32. This trained KAN ran for 20 Epochs.

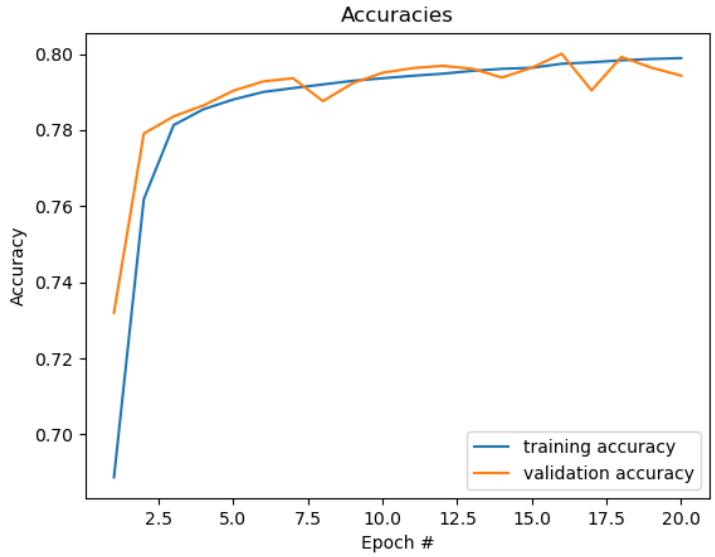

Figure 4.1: The Figure illustrates the progression of KAN_[15,4,2,1] training and validation accuracy over the course of training epochs.

Training accuracy measures how well the model performs on training data and validation accuracy measures how well the KAN performs on unseen data. Figure 4.1 shows, that as the number of epochs increases, the KAN demonstrates improved performance in distinguishing between Farris-type

and Felsenstein-type trees, indicating effective learning from the input data.

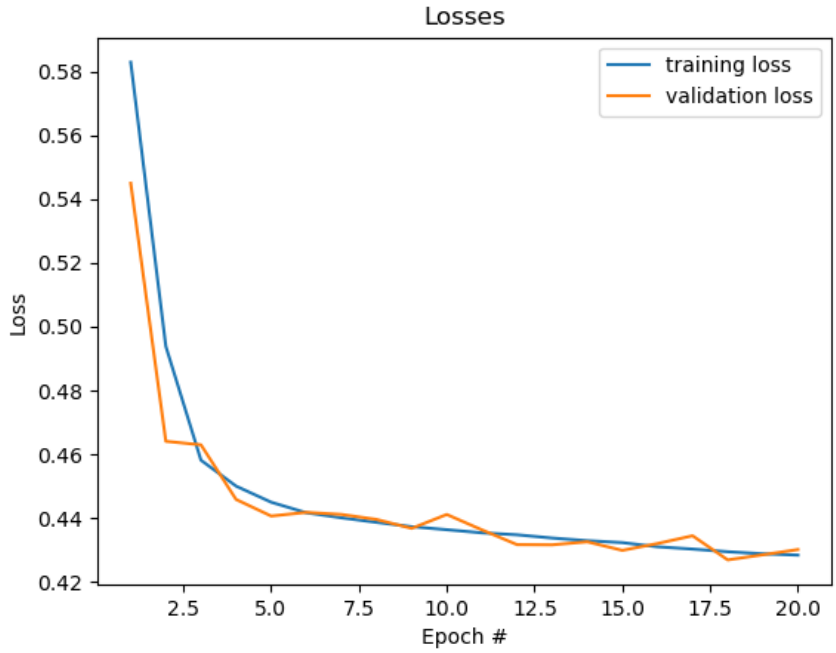

Figure 4.2: This figure presents the corresponding decrease in KAN_[15,4,2,1]'s loss over time.

The reduction in loss, shown in Figure 4.2, reflects the KAN's increasing ability to make accurate predictions, as it minimizes the error between predicted and actual outputs. Together, these figures provide clear evidence that the KAN improves with continued training, both in terms of accuracy and loss, confirming that the network is effectively learning from the data.

The average accuracy for KAN_[15,4,2,1] is 79.58%, the accuracy Felsenstein is 77.37% and the accuracy Farris 81.79%.

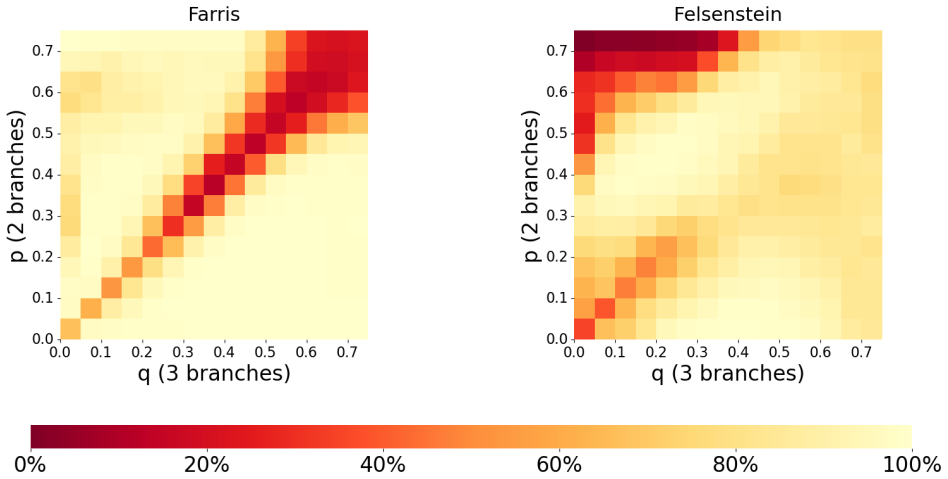

Figure 4.3: The heatmap of KAN_[15,4,2,1] displays the prediction accuracy depending on branch lengths and true tree. It illustrates how accurately the KAN classifies trees, when given an alignment derived from a Farris-type tree (left) or a Felsenstein-type tree (right). The color indicates the percentage of correct classification for the respective (p, q) combinations. The heatmap makes it possible to identify in which branch length ranges the KAN performs particularly efficiently and where the accuracy decreases, revealing important insights into the limitations and robustness of the classification. The upper left corner of both heatmaps corresponds to the Felsenstein zone, i.e., the combinations of (p, q) for which maximum parsimony is statistically inconsistent.

4.3 Accuracies and results for KANs

To evaluate the performance of KANs, multiple hyperparameter configurations were tested. The aim was to understand how variations in network architecture and training settings influence classification accuracy. To isolate the effect of each hyperparameter, one hyperparameter was varied at a time while keeping the others constant. In the first set of experiments, the goal was to optimize accuracy by testing different learning rates, while

keeping the layer structure fixed at $[15, 64, 96, 1]$. The results are shown in Table 4.1.

Table 4.1: Accuracy results for different learning rates (constant size of layers).

Size of layers	**Learning rate**	**Acc Avg in %**	**Acc Far in %**	**Acc Fel in %**
[15, 64, 96, 1]	0.00005	86.67	83.21	90.14
[15, 64, 96, 1]	0.000075	86.80	83.68	89.92
[15, 64, 96, 1]	0.0001	86.84	83.92	89.76
[15, 64, 96, 1]	0.0002	**87.06**	84.88	89.24
[15, 64, 96, 1]	0.0003	86.92	84.02	89.82
[15, 64, 96, 1]	0.0004	87.04	84.99	89.10
[15, 64, 96, 1]	0.0005	86.93	83.87	89.99

Here, all models perform well, with average accuracies ranging from 86.67% to 87.06%. The best accuracy within this group is 87.06%, obtained with a learning rate of 0.0002. On Farris- and Felsenstein-type trees, this configuration reaches 84.88% and 89.24%, respectively. The learning rate does not seem to have a noteworthy impact on the KANs performance.

The second set of experiments focused on exploring different hidden layer configurations, with an emphasis on maintaining a small number of layers to promote interpretability. The goal was to find a simple network structure that still achieves competitive accuracy.

Table 4.2: Comparison of different hyperparameter combinations.

Size of layers	**Learning rate**	**Acc Avg in %**	**Acc Far in %**	**Acc Fel in %**
[15, 5, 1]	0.0002	68.62	63.67	73.57
[15, 3, 1]	0.0002	68.61	63.64	73.59
[15, 50, 1]	0.0002	68.62	63.86	73.37
[15, 100, 1]	0.0002	68.90	60.06	77.74
[15, 4, 2, 1]	0.0002	**82.25**	80.70	83.81
[15, 3, 1]	0.0010	68.53	54.38	82.67
[15, 64, 96, 1]	0.0001	86.84	83.92	89.76
[15, 64, 128, 208, 96, 1]	0.0001	87.68	86.56	88.81
[15, 64, 128, 256, 408, 208, 96, 1]	0.0001	**87.75**	86.41	89.10
[15, 64, 128, 256, 512, 1024, 408, 208, 96, 1]	0.0001	87.63	86.73	88.53

Table 4.2 shows, that a configuration with just two hidden layers $[15, 4, 2, 1]$ reaches an average accuracy of 82.25%, with 80.70% accuracy on Farris-type trees and 83.81% on Felsenstein-type trees. However, models with only a single hidden layer, such as $[15, 3, 1]$ or $[15, 100, 1]$, perform worse, with average accuracies around 68.6–68.9%.

The findings, shown in Table 4.2, suggest that simply increasing the width of a shallow network does not compensate for the representational advantages conferred by additional depth. The architectures with at least two hidden layers achieved significantly better results, providing a more effective trade-off between simplicity and predictive performance.

The best-performing model overall is the one with layer configuration $[15, 64, 128, 256, 408, 208, 96, 1]$ and a learning rate of 0.0001 (see Table 4.2). This model achieves an average accuracy of 87.75%, with 86.41% accuracy for Farris-type trees and 89.10% for Felsenstein-type trees. This model will be referred to as `KAN_best`.

For all KANs presented in Table 4.2, the corresponding heatmaps are provided in Appendix B. These visualizations support the statement that models with only a single hidden layer exhibit lower accuracies compared to more layered models, indicating that shallower architectures are generally less effective.

These experiments demonstrate that while shallow KANs can already achieve respectable accuracies (e.g., 82.25% with just two hidden layers), deeper networks with appropriate learning rates can push performance further, achieving accuracies up to 87.75%. The introduction of even moderate depth proves to be essential for capturing relevant structures in the input

data.

4.4 Comparison of KAN and ANN

To compare the performance of KANs with ANNs, we consider the two best-performing models from each class. The best-performing ANN is the model `F-zoneNN`, as described by Leuchtenberger et al. [2020], which achieved an average accuracy of 87.4%. The best-performing KAN model, referred to as `KAN_best`, as introduced in Section 4.3, achieves a slightly higher average accuracy of 87.75%.

Table 4.3 lists hyperparameters of both models. Apart from the layer architecture, all other hyperparameters are held constant. The ANN uses a deep feedforward structure with ten layers, including a maximum width of 1024 units, while the KAN model uses a slightly shallower architecture with otherwise similar widths.

Table 4.3: Hyperparameters of F-ZoneNN and KAN_best.

Parameter	F-ZoneNN	KAN_best
Size of Layers	[15, 64, 128, 256, 512, 1024, 408, 208, 96, 1]	[15, 64, 128, 256, 408, 208, 96, 1]
Learning Rate	0.0001	0.0001
Batch Size	32	32
Epochs	20	20

Chapter 5

Conclusion and Outlook

5.1 Conclusion

In this thesis, the Kolmogorov-Arnold network architecture was explored and applied to a specific case study. The model achieved strong performance, reaching a classification accuracy of 87.63%, which is comparable to the baseline ANN model presented by Leuchtenberger et al. [2020]. Notably, the KAN was able to reach this level of accuracy using an architecture with fewer layers and nodes. This efficiency highlights the potential of KANs to approximate complex functions with fewer edges.

Even though the KAN demonstrated competitive predictive performance and architectural compactness, the interpretation of the parameters remains difficult. Although the KAN's construction suggests a potential avenue for interpretability in principle, the parameters learned are not straightforward to interpret in biologically meaningful terms. As shown in Chapter 3.4.1 and Appendix A.4, the parameters do not yield simple or immediately inter-

pretable rules for the phylogenetic task discussed in this thesis.

While the KAN architecture provides a compact and competitive model for the phylogenetic classification problem studied here, this evaluation is based on a single use case. In order to make broader claims about the general advantages of KANs over traditional neural network architectures, further research is required. Future work should include testing across a wider range of datasets. The work presented here therefore highlights a balanced picture: KANs appear promising as compact, effective models for the studied task, but additional application would be necessary before claiming interpretability advantages or general superiority over conventional architectures.

5.2 Outlook

While the Kolmogorov-Arnold network has demonstrated promising results in the phylogenetic application presented in this thesis, several important directions for future research remain open. The generalizability of KANs should be investigated more thoroughly. This includes testing on larger and more diverse datasets and comparing performance not only in terms of accuracy but also computational efficiency, training time, and stability. Further analysis of the interpretability of KANs could help clarify whether the theoretical advantages of their structure translate into practical insights.

Benchmarking KANs against state-of-the-art models in real-world applications will be essential to determine whether they represent a meaningful advancement over existing methods, or if their benefits are primarily theoretical or case-dependent.

Appendix A

B-spline Calculations

B-splines on the grid

$$t_i = \left\{ -\frac{11}{5}, -\frac{9}{5}, -\frac{7}{5}, -1, -\frac{3}{5}, -\frac{1}{5}, \frac{1}{5}, \frac{3}{5}, 1, \frac{7}{5}, \frac{9}{5}, \frac{11}{5} \right\}$$

A.1 Degree 0

$$B_{0,0}(x) = \begin{cases} 1 & \text{if } x \geq -\frac{11}{5} \wedge x < -\frac{9}{5} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{1,0}(x) = \begin{cases} 1 & \text{if } x \geq -\frac{9}{5} \wedge x < -\frac{7}{5} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{2,0}(x) = \begin{cases} 1 & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ 0 & \text{otherwise} \end{cases}$$

$$B_{3,0}(x) = \begin{cases} 1 & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{4,0}(x) = \begin{cases} 1 & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{5,0}(x) = \begin{cases} 1 & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{6,0}(x) = \begin{cases} 1 & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{7,0}(x) = \begin{cases} 1 & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$B_{8,0}(x) = \begin{cases} 1 & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{9,0}(x) = \begin{cases} 1 & \text{if } x \geq \frac{7}{5} \wedge x < \frac{9}{5} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{10,0}(x) = \begin{cases} 1 & \text{if } x \geq \frac{9}{5} \wedge x < \frac{11}{5} \\ 0 & \text{otherwise} \end{cases}$$

A.2 Degree 1

$$B_{0,1}(x) = \begin{cases} \frac{5x+11}{2} & \text{if } x \geq -\frac{11}{5} \wedge x < -\frac{9}{5} \\ -\frac{5x+7}{2} & \text{if } x \geq -\frac{9}{5} \wedge x < -\frac{7}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{1,1}(x) = \begin{cases} \frac{5x+9}{2} & \text{if } x \geq -\frac{9}{5} \wedge x < -\frac{7}{5} \\ -\frac{5(x+1)}{2} & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{2,1}(x) = \begin{cases} \frac{5x+7}{2} & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ -\frac{5x+3}{2} & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{3,1}(x) = \begin{cases} -\frac{5x+1}{2} & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ \frac{5(x+1)}{2} & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{4,1}(x) = \begin{cases} \frac{5x+3}{2} & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ \frac{1-5x}{2} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{5,1}(x) = \begin{cases} \frac{5x+1}{2} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ \frac{3-5x}{2} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{6,1}(x) = \begin{cases} \frac{5x-1}{2} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ \frac{5(1-x)}{2} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{7,1}(x) = \begin{cases} \frac{5x-3}{2} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ \frac{7-5x}{2} & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{8,1}(x) = \begin{cases} \frac{9-5x}{2} & \text{if } x \geq \frac{7}{5} \wedge x < \frac{9}{5} \\ \frac{5(x-1)}{2} & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{9,1}(x) = \begin{cases} \frac{5x-7}{2} & \text{if } x \geq \frac{7}{5} \wedge x < \frac{9}{5} \\ \frac{11-5x}{2} & \text{if } x \geq \frac{9}{5} \wedge x < \frac{11}{5} \\ 0 & \text{if otherwise} \end{cases}$$

A.3 Degree 2

$$B_{0,2}(x) = \begin{cases} \frac{(5x+11)^2}{8} & \text{if } x \geq -\frac{11}{5} \wedge x < -\frac{9}{5} \\ -(\frac{25x^2}{4} + 20x + \frac{61}{4}) & \text{if } x \geq -\frac{9}{5} \wedge x < -\frac{7}{5} \\ \frac{25(x+1)^2}{8} & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{1,2}(x) = \begin{cases} -(\frac{25x^2}{4} + 15x + \frac{33}{4}) & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ \frac{(5x+3)^2}{8} & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ \frac{(5x+9)^2}{8} & \text{if } x \geq -\frac{9}{5} \wedge x < -\frac{7}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{2,2}(x) = \begin{cases} \frac{(5x+1)^2}{8} & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ -(\frac{25x^2}{4} + 10x + \frac{13}{4}) & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ \frac{(5x+7)^2}{8} & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{3,2}(x) = \begin{cases} -(\frac{25x^2}{4} + 5x + \frac{1}{4}) & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ \frac{(5x-1)^2}{8} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ \frac{25(x+1)^2}{8} & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{4,2}(x) = \begin{cases} \frac{3-25x^2}{4} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ \frac{(5x-3)^2}{8} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ \frac{(5x+3)^2}{8} & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{5,2}(x) = \begin{cases} \frac{(5x+1)^2}{8} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ -\frac{25x^2}{4} + 5x - \frac{1}{4} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ \frac{25(x-1)^2}{8} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{6,2}(x) = \begin{cases} \frac{(5x-1)^2}{8} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ -\frac{25x^2}{4} + 10x - \frac{13}{4} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ \frac{(5x-7)^2}{8} & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{7,2}(x) = \begin{cases} \frac{(5x-3)^2}{8} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ -\frac{25x^2}{4} + 15x - \frac{33}{4} & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ \frac{(5x-9)^2}{8} & \text{if } x \geq \frac{7}{5} \wedge x < \frac{9}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{8,2}(x) = \begin{cases} -\frac{25x^2}{4} + 20x - \frac{61}{4} & \text{if } x \geq \frac{7}{5} \wedge x < \frac{9}{5} \\ \frac{(5x-11)^2}{8} & \text{if } x \geq \frac{9}{5} \wedge x < \frac{11}{5} \\ \frac{25(x-1)^2}{8} & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ 0 & \text{if otherwise} \end{cases}$$

A.4 Degree 3

$$B_{0,3}(x) = \begin{cases} \frac{(5x+11)^3}{48} & \text{if } x \geq -\frac{11}{5} \wedge x < -\frac{9}{5} \\ -\frac{5\left(75x^3+375x^2+609x+317\right)}{48} & \text{if } x \geq -\frac{9}{5} \wedge x < -\frac{7}{5} \\ \frac{375x^3+1275x^2+1365x+473}{48} & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ -\frac{(5x+3)^3}{48} & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{1,3}(x) = \begin{cases} -\frac{375x^3+1425x^2+1725x+643}{48} & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ \frac{375x^3+825x^2+525x+107}{48} & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ \frac{(5x+9)^3}{48} & \text{if } x \geq -\frac{9}{5} \wedge x < -\frac{7}{5} \\ -\frac{(5x+1)^3}{48} & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{2,3}(x) = \begin{cases} \frac{5\left(75x^3+75x^2+9x+1\right)}{48} & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ -\frac{375x^3+975x^2+765x+157}{48} & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ \frac{(5x+7)^3}{48} & \text{if } x \geq -\frac{7}{5} \wedge x < -1 \\ -\frac{(5x-1)^3}{48} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{3,3}(x) = \begin{cases} \frac{375x^3-75x^2-75x+23}{48} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ -\frac{(5x-3)^3}{48} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ \frac{-375x^3-525x^2-165x+17}{48} & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ \frac{125(x+1)^3}{48} & \text{if } x \geq -1 \wedge x < -\frac{3}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{4,3}(x) = \begin{cases} \frac{-375x^3-75x^2+75x+23}{48} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ \frac{375x^3-525x^2+165x+17}{48} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ \frac{(5x+3)^3}{48} & \text{if } x \geq -\frac{3}{5} \wedge x < -\frac{1}{5} \\ -\frac{125(x-1)^3}{48} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{5,3}(x) = \begin{cases} \frac{(5x+1)^3}{48} & \text{if } x \geq -\frac{1}{5} \wedge x < \frac{1}{5} \\ \frac{5\left(-75x^3+75x^2-9x+1\right)}{48} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ \frac{375x^3-975x^2+765x-157}{48} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ -\frac{(5x-7)^3}{48} & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{6,3}(x) = \begin{cases} \frac{(5x-1)^3}{48} & \text{if } x \geq \frac{1}{5} \wedge x < \frac{3}{5} \\ \frac{-375x^3+825x^2-525x+107}{48} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ \frac{375x^3-1425x^2+1725x-643}{48} & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ -\frac{(5x-9)^3}{48} & \text{if } x \geq \frac{7}{5} \wedge x < \frac{9}{5} \\ 0 & \text{if otherwise} \end{cases}$$

$$B_{7,3}(x) = \begin{cases} \frac{(5x-3)^3}{48} & \text{if } x \geq \frac{3}{5} \wedge x < 1 \\ \frac{-375x^3+1275x^2-1365x+473}{48} & \text{if } x \geq 1 \wedge x < \frac{7}{5} \\ \frac{5\left(75x^3-375x^2+609x-317\right)}{48} & \text{if } x \geq \frac{7}{5} \wedge x < \frac{9}{5} \\ -\frac{(5x-11)^3}{48} & \text{if } x \geq \frac{9}{5} \wedge x < \frac{11}{5} \\ 0 & \text{if otherwise} \end{cases}$$

Appendix B

Additional results of KAN training

B.1 KAN_[15,64,128,256,512,1024,408,208,96,1]

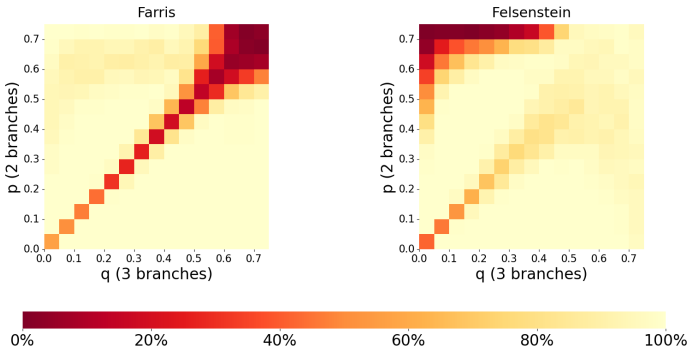

Figure B.1: Heatmap of KAN_[15,64,128,256,512,1024,408,208,96,1]

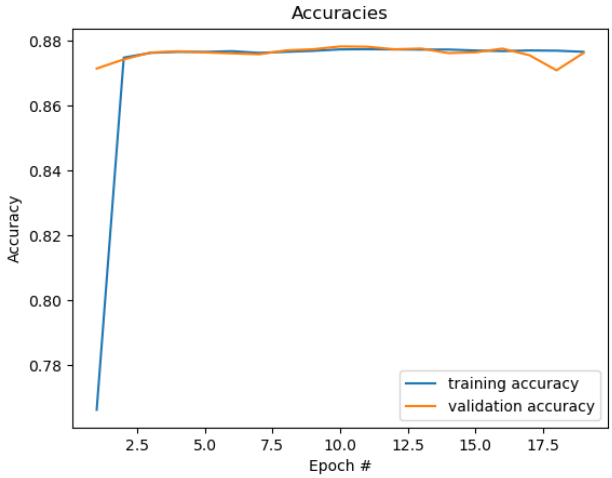

Figure B.2: Accuracy of KAN_[15,64,128,256,512,1024,408,208,96,1]

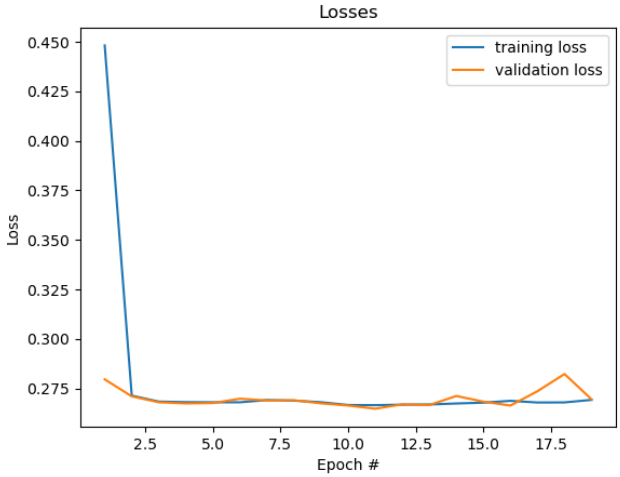

Figure B.3: Loss of KAN_[15,64,128,256,512,1024,408,208,96,1]

B.2 KAN_[15,64,96,1]

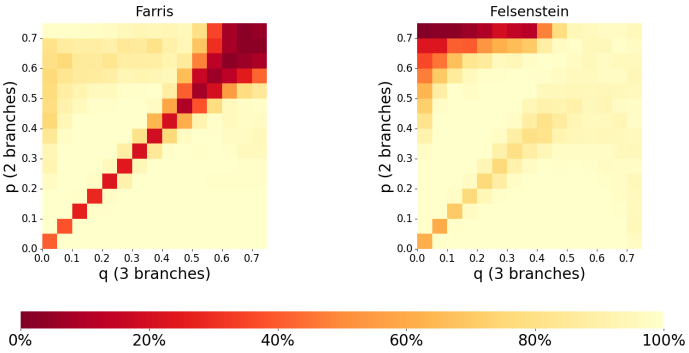

Figure B.4: Heatmap of KAN_[15,64,96,1]

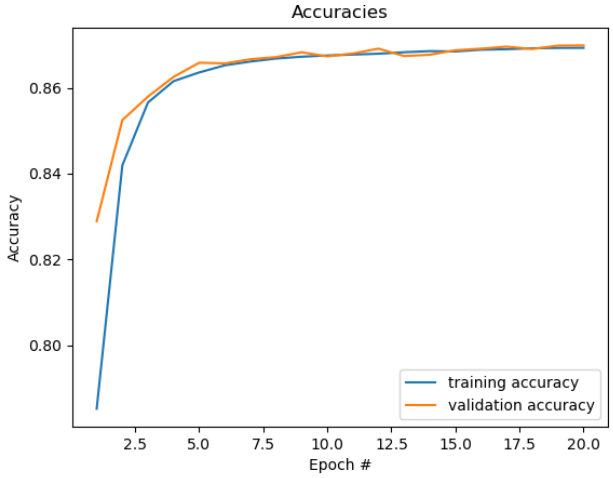

Figure B.5: Accuracy of KAN_[15,64,96,1]

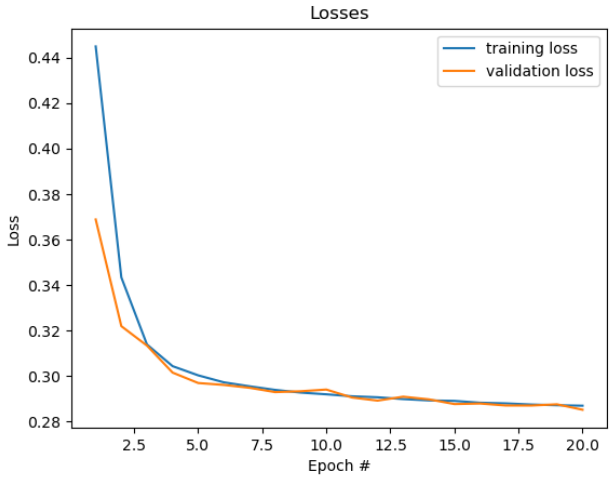

Figure B.6: Loss of KAN_[15,64,96,1]

B.3 KAN_best ([15,64,128,256,408,208,96,1])

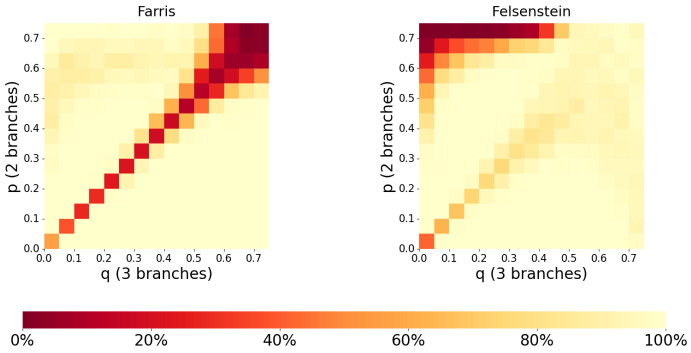

Figure B.7: Heatmap of KAN_bast ([15,64,128,256,408,208,96,1])

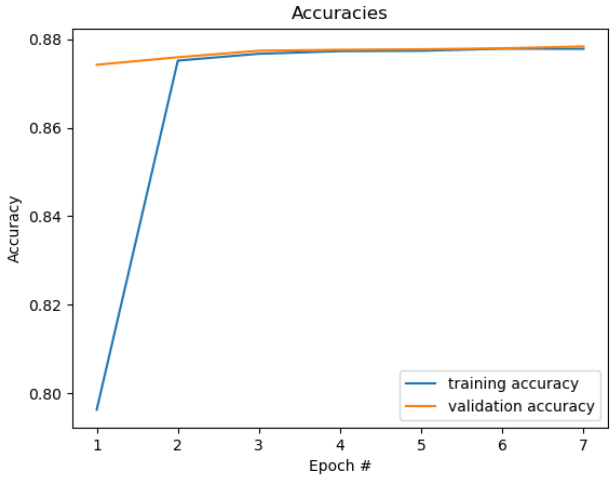

Figure B.8: Accuracy of KAN_[15,64,128,256,408,208,96,1]

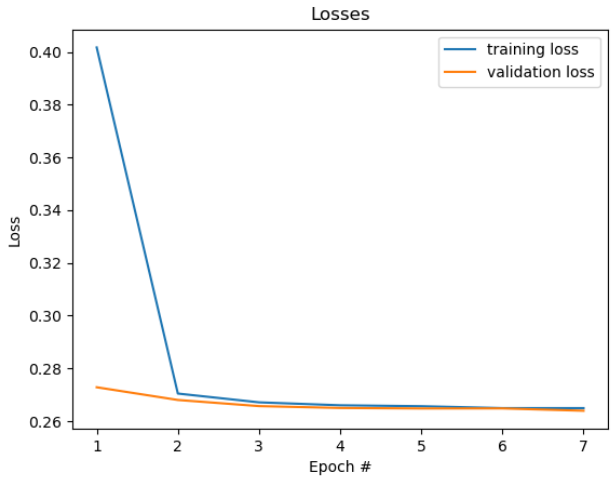

Figure B.9: Loss of KAN_[15,64,128,256,408,208,96,1]

B.4 KAN_[15,64,128,208,96,1]

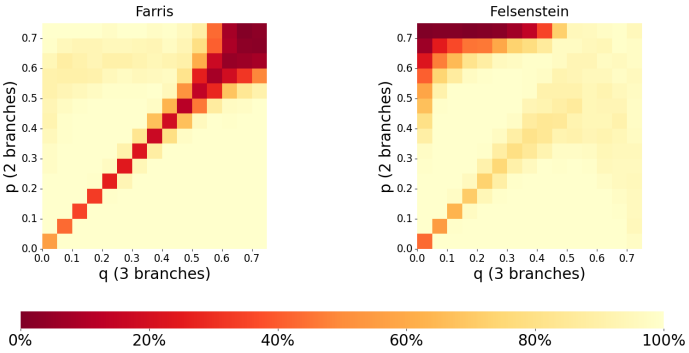

Figure B.10: Heatmap of KAN_[15,64,128,208,96,1]

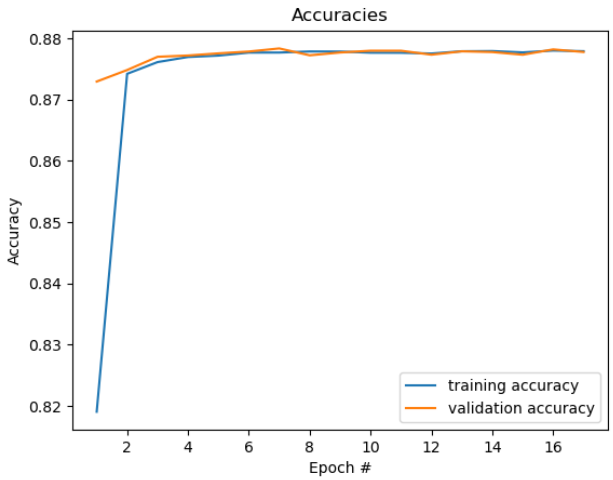

Figure B.11: Accuracy of KAN_[15,64,128,208,96,1]

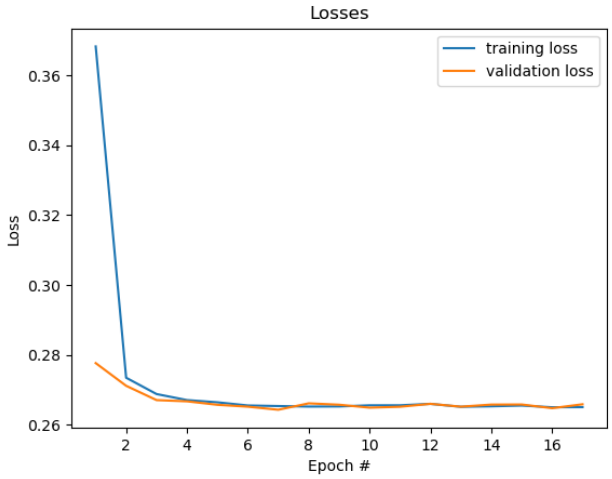

Figure B.12: Loss of KAN_[15,64,128,208,96,1]

B.5 KAN_[15,5,1]

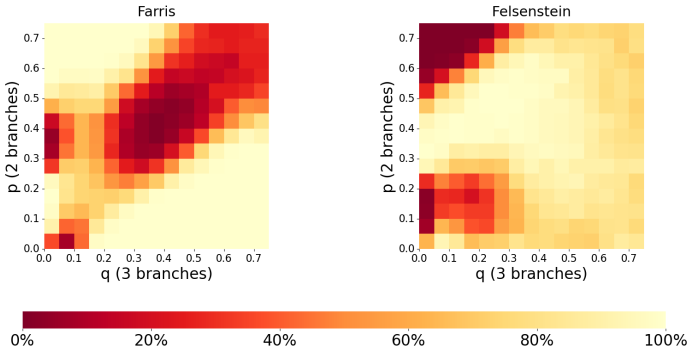

Figure B.13: Heatmap of KAN_[15,5,1]

B.6 KAN_[15,3,1], learning rate 0.0002

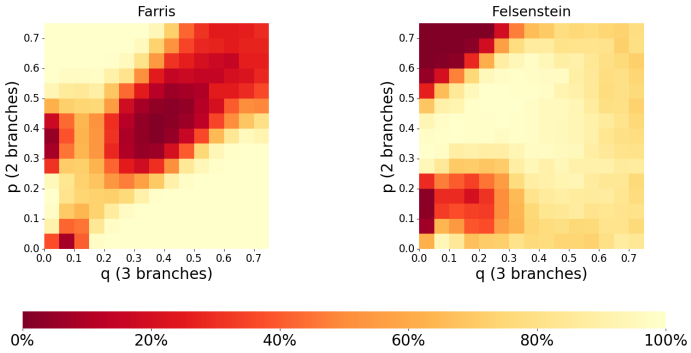

Figure B.14: Heatmap of KAN_[15,3,1], learning rate 0.0002

B.7 KAN_[15,3,1], learning rate 0.001

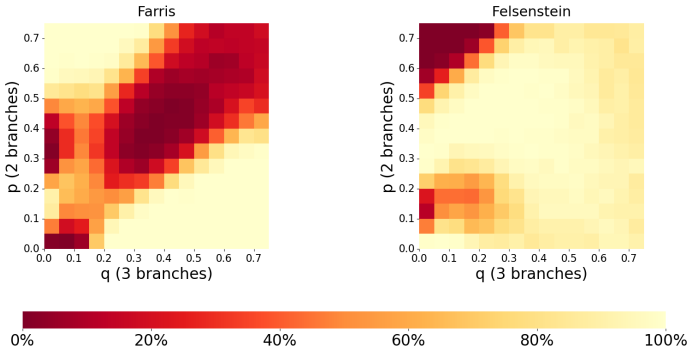

Figure B.15: Heatmap of KAN_[15,3,1], learning rate 0.001

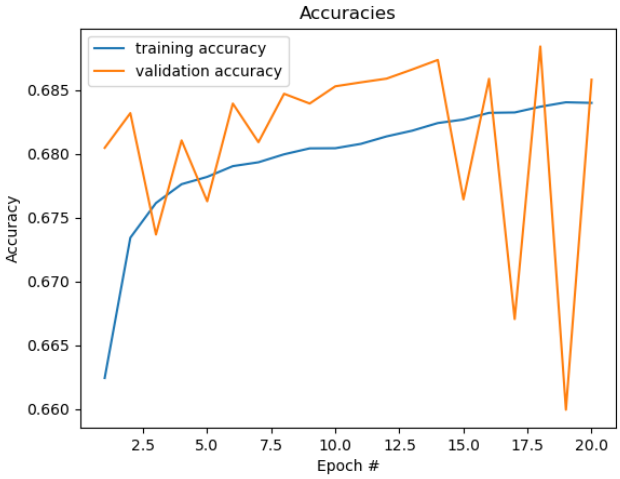

Figure B.16: Accuracy of KAN_[15,3,1], learning rate 0.001

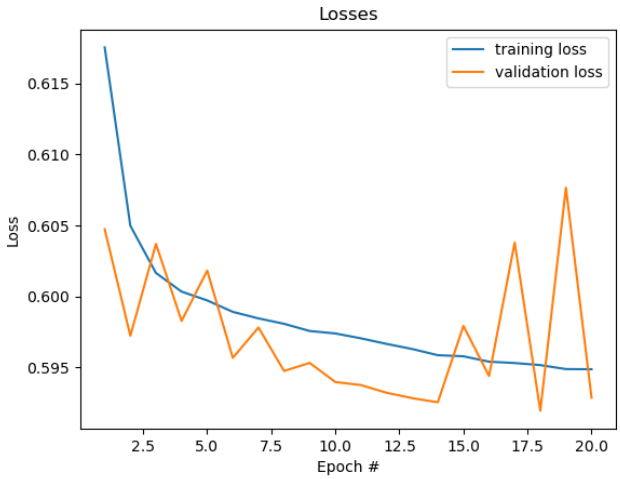

Figure B.17: Loss of KAN_[15,3,1], learning rate 0.001

B.8 KAN_[15,4,2,1]

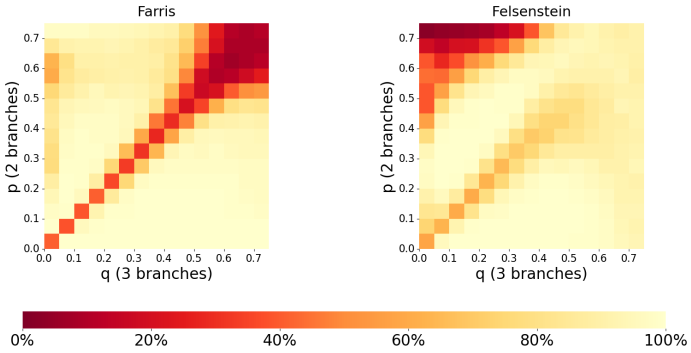

Figure B.18: Heatmap of KAN_[15,4,2,1]

B.9 KAN_[15,50,1]

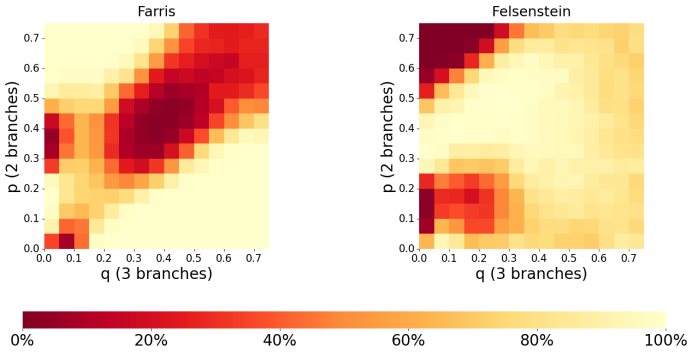

Figure B.19: Heatmap of KAN_[15,50,1]

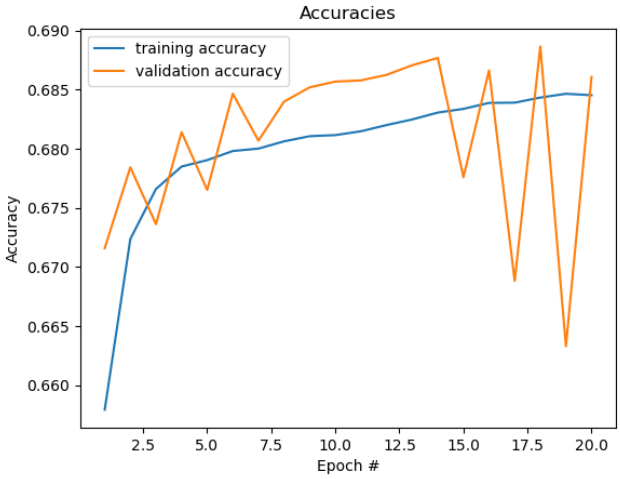

Figure B.20: Accuracy of KAN_[15,50,1]

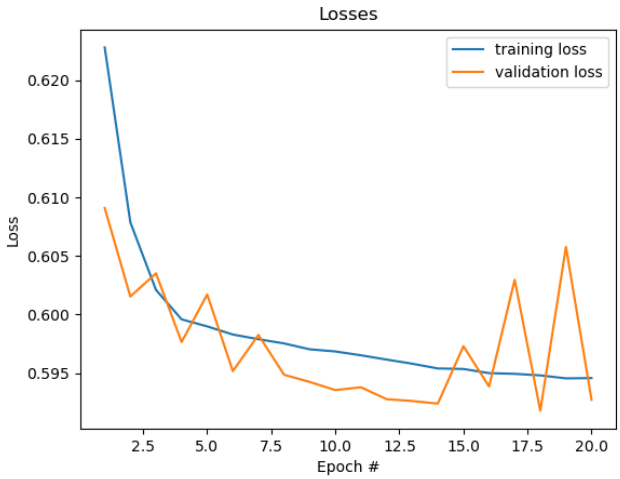

Figure B.21: Loss of KAN_[15,50,1]

B.10 KAN_[15,100,1]

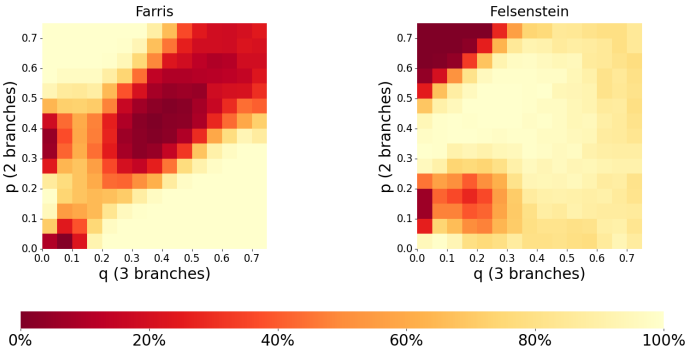

Figure B.22: Heatmap of KAN_[15,100,1]

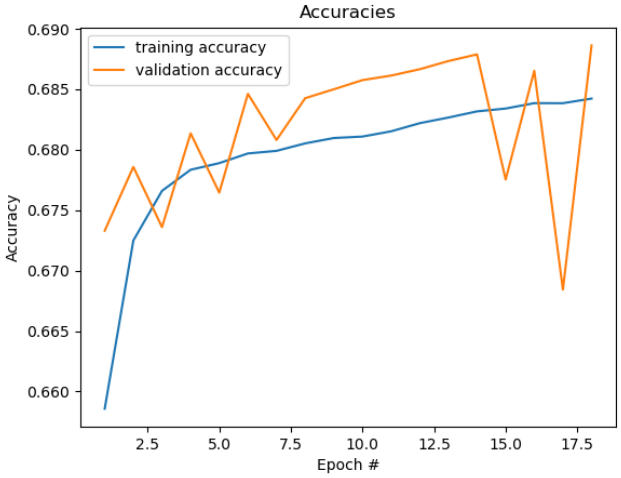

Figure B.23: Accuracy of KAN_[15,100,1]

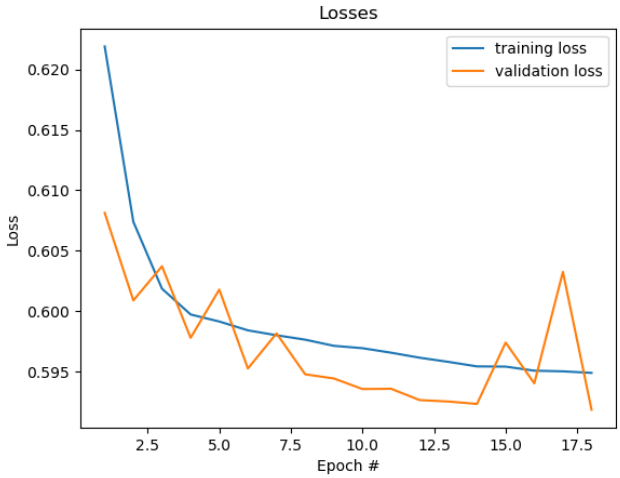

Figure B.24: Loss of KAN_[15,100,1]

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. arXiv preprint arXiv:1206.5533v2, 2012. version 2 submitted on September 16, 2012.

François Chollet et al. Keras. `https://keras.io`, 2015.

Henrik Christensen. *Einführung in die Bioinformatik in der Mikrobiologie.* Springer Spektrum, 2023.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Systems*, 2:303–314, 1989. doi: 10.1007/BF02551274.

Carl de Boor. On calculating with b-splines. *Journal of Approximation Theory*, 6:50–62, 1972. doi: 10.1016/0021-9045(72)90080-9.

Carl de Boor. *A Practical Guide to Splines*. Applied Mathematical Sciences. Springer-Verlag, 2001. ISBN 9780387953663.

A De Bruyn, DP Martin, and P. Lefeuvre. Phylogenetic reconstruction methods: an overview. 2014. doi: 10.1007/978-1-62703-767-9_13.

A. W. F. Edwards and L. L. Cavalli-Sforza. Reconstruction of evolutionary trees. In V. H. Heywood and J. McNeill, editors, *Phenetic and Phylogenetic Classification*, page 67–76. Systematics Association Publ. No.6, London, England, 1964.

Joseph Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. *Syst. Biol.*, 27(4):401–410, 1978. doi: 10.1093/sysbio/27.4.401.

Joseph Felsenstein. Evolutionary trees from dna sequences: A maximum likelihood approach. *J. Mol. Evol.*, 17(6):368–376, nov 1981. ISSN 1432-1432. doi: 10.1007/BF01734359.

Joseph Felsenstein. *Inferring phylogenies*. Sinauer Associates, Inc., Sunderland (MA), USA, 2004. ISBN 0878931775.

Walter M. Fitch. Toward defining the course of evolution: Minimum change

for a specific tree topology. *Syst. Zool.*, 20(4):406–416, 1971. ISSN 00397989. doi: 10.2307/2412116.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Matt Haber and Joel Velasco. Phylogenetic inference. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2024 edition, 2024. `https://plato.stanford.edu/archives/sum2024/entries/phylogenetic-inference/`.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Pergamon Press*, 4:303–314, 1991. doi: 10.1016/0893-6080(91)90009-T.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8.

Thomas H Jukes and Charles R Cantor. Evolution of protein molecules. In Hamish Nisbet Munro, editor, *Mammalian Protein Metabolism*, pages 21–132. Academic Press, New York (NY), USA, 1969. doi: 10.1016/B978-1-4832-3211-9.50009-7.

A.N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114(5), 1957.

Alina F. Leuchtenberger, Stephen M. Crotty, Tamara Drucks, Heiko A. Schmidt, Sebastian Burgstaller-Muehlbacher, and Arndt von Haeseler.

Distinguishing felsenstein zone from farris zone using neural networks. *Molecular Biology and Evolution*, 37(12):3632–3641, 2020. doi: 10.1093/molbev/msaa164.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks, 2024. Accepted at ICLR 2025.

Horst D. Lohrer. *Einführung in die Bioinformatik*. Springer Spektrum, 2022.

Tom M. Mitchell. *Machine Learning*. WCB McGraw–Hill, 1997. ISBN 978-0-07-042807-2.

Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

Les Piegl and Wayne Tiller. *The NURBS Book*. Springer, Berlin, Heidelberg, 2 edition, 1997. ISBN 978-3-540-61545-3. doi: 10.1007/978-3-642-59223-2.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi: 10.1017/S0962492900002919.

Andrew Rambaut and Nicholas C. Grass. Seq-Gen: an application for the monte carlo simulation of dna sequence evolution along phylogenetic trees. *Bioinformatics*, 13(3):235–238, jun 1997. ISSN 1367-4803. doi: 10.1093/bioinformatics/13.3.235.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.

David Sankoff. Minimal mutation trees of sequences. *SIAM J. Appl. Math.*, 28(1):35–42, 1975. ISSN 00361399. URL `http://www.jstor.org/stable/2100459`.

Ajay Shrestha and Ausif Mahmood. Review of deep learning algorithms and architectures. *IEEE Access*, 7:53040–53065, 2019. doi: 10.1109/ACCESS.2019.2912200.

Mark E Siddall. Success of parsimony in the four-taxon case: Long-Branch repulsion by likelihood in the farris zone. *Cladistics*, 14(3):209–220, 1998. doi: 10.1006/clad.1998.0063.

Peter J Waddell. *Statistical methods of phylogenetic analysis: including Hadamard conjugations, LogDet transforms and maximum likelihood*. PhD thesis, Massey University, Palmerston North, New Zealand, 1995.

Z Yang. Statistical properties of the maximum likelihood method of phylogenetic estimation and comparison with distance matrix methods. *Syst. Biol.*, 43(3):329–342, 1994. doi: 10.1093/sysbio/43.3.329.

Zhipeng Zhou. tfkan, tensorflow implementation for kan. `https://github.com/ZPZhou-lab/tfkan`, 2024. MIT License.