Neural Tangent Kernels for Axis-Aligned Tree Ensembles

Ryuichi Kanoh [1 2] Mahito Sugiyama [1 2]

Abstract

While *axis-aligned rules* are known to induce an important inductive bias in machine learning models such as typical hard decision tree ensembles, theoretical understanding of the learning behavior is largely unrevealed due to the discrete nature of rules. To address this issue, we impose the axis-aligned constraint on *soft trees*, which relax the splitting process of decision trees and are trained using a gradient method, and present their *Neural Tangent Kernel* (NTK), which enables us to analytically describe the training behavior. We study two cases: imposing the axis-aligned constraint throughout the entire training process, and only at the initial state. Moreover, we extend the NTK framework to handle various tree architectures simultaneously, and prove that any axis-aligned non-oblivious tree ensemble can be transformed into axis-aligned oblivious tree ensembles with the same NTK. One can search for suitable tree architecture via Multiple Kernel Learning (MKL), and our numerical experiments show a variety of suitable features depending on the type of constraints. Our NTK analysis highlights both the theoretical and practical impacts of the axis-aligned constraint in tree ensemble learning.

1. Introduction

One of the most practical machine learning techniques used in real-world applications is *ensemble learning*. It combines the outputs of multiple predictors, often referred to as weak learners, to obtain reliable results for complex prediction problems. A hard decision tree is commonly used as a weak learner. Its inductive bias caused by the *axis-aligned splitting* of a feature space is considered to be important. For example, Grinsztajn et al. (2022) experimentally demonstrated that the presence or absence of rotational invariance due to the axis-aligned constraint has a significant impact on generalization performance, especially for tabular datasets. However, although a number of machine learning models have been proposed that are aware of axis-aligned partitioning (Chang et al., 2022; Humbird et al., 2019), it has not been theoretically clear what properties emerge when the axis-aligned constraints are imposed.

In this paper, we consider ensemble learning of *soft trees* to realize a differentiable analysis of axis-aligned splitting. The concept of a soft tree is characterized by its ability to adjust the parameters of the entire model through gradient-based optimization. Ensembles of soft trees are recognized for their high empirical performance (Kontschieder et al., 2015; Popov et al., 2020; Hazimeh et al., 2020). Furthermore, the differentiability of soft trees allows for integration with various deep learning methodologies, such as fine-tuning (Ke et al., 2019), pre-training (Arik & Pfister, 2021), dropout (Srivastava et al., 2014), and various stochastic gradient descent methods (Kingma & Ba, 2015; Foret et al., 2021), resulting in desirable traits in real-world settings. Another advantage of soft trees is their interpretability (Frosst & Hinton, 2017; Chang et al., 2022). Soft trees have also been implemented in well-known open-source software such as PyTorch Tabular (Joseph, 2021), and their practical application is thriving.

Recently, there has been progress in the theoretical analysis of soft tree ensembles (Kanoh & Sugiyama, 2022; 2023) using the *Neural Tangent Kernel* (NTK) framework (Jacot et al., 2018), which provides analytical descriptions of ensemble learning with infinitely many soft trees, yielding several non-trivial properties such as the existence of tree architectures that have exactly the same training behavior even if they are non-isomorphic. However, the current NTK analysis assumes all the input features to be taken into account in each splitting process of soft trees, resulting in oblique splitting boundaries. Therefore, it cannot directly incorporate the axis-aligned constraint in its current state.

In this paper, we extend the NTK concept to the axis-aligned soft tree ensembles to uncover the theoretical properties of the axis-aligned constraint. We have revised the basic training methodology and gained several insights that go beyond mere extensions of Kanoh & Sugiyama (2022; 2023) through examining the obtained novel kernels.

[1]National Institute of Informatics [2]The Graduate University for Advanced Studies, SOKENDAI. Correspondence to: Ryuichi Kanoh <kanoh@nii.ac.jp>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Our contributions can be summarized as follows:

- **Closed-form NTK induced by two axis-aligned training formulation.**

  We succeed in extending the prior work of Kanoh & Sugiyama (2022; 2023) to axis-aligned trees and successfully formulate a closed-form kernel induced by infinitely many axis-aligned tree ensembles. Using this kernel, we are able to analytically obtain the behavior of the training of axis-aligned trees (Theorem 3.1). We conducted a theoretical analysis on two cases: one in which the axis-aligned constraint is always imposed during training, and the other in which only the initial model is axis-aligned and training is conducted freely from there (Figure 1).

- **Deriving the NTK of ensembles of various tree architectures.**

  Prior studies (Kanoh & Sugiyama, 2022; 2023) were limited to the scenario of an infinite number of weak learners with identical architectures, which is not practical. This limitation is particularly unrealistic for axis-aligned trees, where a single feature is used at each split (for example, the second feature is always used at the first node in all trees, and the third feature is not used at all). This may cause a lack of representation power. We successfully address this limitation by decomposing the NTK induced by a model into the sum of the NTKs induced by its sub-models (Proposition 3.1), which is applicable to any ensemble models such as Generalized Additive Models (GAM) (Hastie & Tibshirani, 1986).

- **Sufficiency of the oblivious tree for architecture search.** The oblivious tree is a perfect binary tree with shared splitting rules at each depth, resulting in efficient computation. We show that any axis-aligned non-oblivious tree ensemble, including those with non-perfect binary tree architectures, can be transformed into a set of axis-aligned oblivious tree ensembles that induce exactly the same NTK (Proposition 4.1). This proposition enables us to substantially reduce the number of potential tree architecture patterns.

- **Finding suitable tree architecture via Multiple Kernel Learning (MKL).**

  We employ MKL (Gönen & Alpaydın, 2011; Aiolli & Donini, 2015), which determines the weights of a linear combination of multiple kernels during training, to analyze the effect of the axis-aligned constraint in feature selection. The learned weights of the linear combination of NTKs induced by various tree architectures can be interpreted, using Proposition 3.1, as the proportion of the presence of each tree architecture. Our empirical experiments suggest that the suitable features vary depending on the type of training constraints. We empirically investigate the properties of the axis-aligned tree models with respect to the type of constraints (axis-aligned or oblique) and training (gradient descent or greedy search) via our NTK.

2. Preliminaries

This section covers the introduction of the soft tree ensemble and the NTK.

2.1. Soft Tree Ensembles

We formulate regression using soft decision trees based on the literature (Kontschieder et al., 2015). Let $\boldsymbol{x} \in \mathbb{R}^{F \times N}$ be input data consisting of N samples with F features. Assume that there are M soft decision trees and each tree has $\mathcal{N}$ splitting nodes and $\mathcal{L}$ leaf nodes. For each tree $m \in [M] = \{1, \dots, M\}$, we denote trainable parameters of the m-th soft decision tree as $\boldsymbol{w}_m \in \mathbb{R}^{F \times \mathcal{N}}$ and $\boldsymbol{b}_m \in \mathbb{R}^{1 \times \mathcal{N}}$ for splitting nodes, which correspond to feature selection and splitting threshold in typical decision trees, and $\boldsymbol{\pi}_m \in \mathbb{R}^{1 \times \mathcal{L}}$ for leaf nodes.

Unlike typical hard decision trees, each leaf node $\ell \in [\mathcal{L}] = \{1, \dots, \mathcal{L}\}$ in soft decision trees holds a value $\mu_{m,\ell}(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{b}_m) \in [0,1]$ that represents the probability of input data reaching the leaf ℓ:

$$\mu_{m,\ell}(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{b}_m) = \prod_{n=1}^{\mathcal{N}} \underbrace{\sigma(\boldsymbol{w}_{m,n}^\top \boldsymbol{x}_i + \beta b_{m,n})^{\mathbb{1}_{\ell \swarrow n}}}_{\text{flow to the left}} \underbrace{(1 - \sigma(\boldsymbol{w}_{m,n}^\top \boldsymbol{x}_i + \beta b_{m,n}))^{\mathbb{1}_{n \searrow \ell}}}_{\text{flow to the right}}, \tag{1}$$

where $\boldsymbol{w}_{m,n} \in \mathbb{R}^F$ and $b_{m,n} \in \mathbb{R}$ are splitting node parameters for an n-th node in an m-th tree. $\beta \in \mathbb{R}^+$ is a hyperparameter that adjusts the influence of the splitting threshold, and $\mathbb{1}_{\ell \swarrow n}(\mathbb{1}_{n \searrow \ell})$ is an indicator function that returns 1 if the ℓ-th leaf is on the left (right) side of a node n and 0 otherwise. Internal nodes use a decision function $\sigma : \mathbb{R} \to [0,1]$ that resembles the sigmoid. This function is rotationally symmetric about the point $(0, 1/2)$ and satisfies the conditions: $\lim_{c \to \infty} \sigma(c) = 1$, $\lim_{c \to -\infty} \sigma(c) = 0$, and $\sigma(0) = 0.5$ for $c \in \mathbb{R}$. Examples of such functions include the two-class sparsemax function given as $\sigma(c) = \mathrm{sparsemax}([\alpha c, 0])$ (Martins & Astudillo, 2016), and the two-class entmax function given as $\sigma(c) = \mathrm{entmax}([\alpha c, 0])$ (Peters et al., 2019). In this paper, we mainly consider the scaled error function: $\sigma(c) = \frac{1}{2}\mathrm{erf}(\alpha c) + \frac{1}{2} = \frac{1}{2}\left(\frac{2}{\sqrt{\pi}} \int_0^{\alpha c} e^{-t^2}\,\mathrm{d}t\right) + \frac{1}{2}$. As the scaling factor $\alpha \in \mathbb{R}^+$ (Frosst & Hinton, 2017) tends towards infinity, sigmoid-like decision functions become step functions that correspond to (hard) Boolean operation. If we replace the right-flow term $(1 - \sigma(\boldsymbol{w}_{m,n}^\top \boldsymbol{x}_i + \beta b_{m,n}))$ with

0 in Equation (1), we obtain a rule set, which is represented by a linear graph architecture.

It is possible to handle both continuous and categorical variables within the same formulation. The most straightforward approach for categorical variables is to use One-Hot encoding. Additionally, previous studies often employed transformed categorical variables using target encoding (Popov et al., 2020; Chang et al., 2022). These procedures can be naturally applied as is, even in the axis-aligned splitting that will be formulated later.

The function $f_m : \mathbb{R}^F \times \mathbb{R}^{F\times\mathcal{N}} \times \mathbb{R}^{1\times\mathcal{N}} \times \mathbb{R}^{1\times\mathcal{L}} \to \mathbb{R}$ that returns the prediction of the m-th tree is given by the weighted sum of leaf-specific parameters $\pi_{m,\ell} \in \mathbb{R}$, weighted by the probability that the input data $\boldsymbol{x}_i$ reaches each leaf:

$$f_m(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{b}_m, \boldsymbol{\pi}_m) = \sum_{\ell=1}^{\mathcal{L}} \pi_{m,\ell}\mu_{m,\ell}(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{b}_m). \quad (2)$$

Furthermore, the output of the ensemble model with M trees for each input $\boldsymbol{x}_i$ is formulated as a function $f : \mathbb{R}^F \times \mathbb{R}^{M\times F\times\mathcal{N}} \times \mathbb{R}^{M\times 1\times\mathcal{N}} \times \mathbb{R}^{M\times 1\times\mathcal{L}} \to \mathbb{R}$ as follows:

$$f(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{b}, \boldsymbol{\pi}) = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} f_m(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{b}_m, \boldsymbol{\pi}_m). \quad (3)$$

In general, parameters $\boldsymbol{w} = (\boldsymbol{w}_1, \dots, \boldsymbol{w}_M)$, $\boldsymbol{b} = (\boldsymbol{b}_1, \dots, \boldsymbol{b}_M)$, and $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_M)$ are randomly initialized using independently and identically distributed normal distributions with mean 0 and variance 1 and updated using gradient descent. In this paper, the term "tree architecture" refers to both the graph topological structure of the tree and a parameter initialization method at each node. Even if the graph topology is identical, those architectures are considered to be distinct when different parameter initialization methods are adopted.

2.2. Neural Tangent Kernels

We introduce the NTK based on the gradient flow using training data $\boldsymbol{x} \in \mathbb{R}^{F\times N}$, the prediction target $\boldsymbol{y} \in \mathbb{R}^N$, trainable parameters $\boldsymbol{\theta}_\tau \in \mathbb{R}^P$ at time τ, and an arbitrary model function $g(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau) : \mathbb{R}^F \times \mathbb{R}^P \to \mathbb{R}$. With the learning rate η and the mean squared error loss function L, the gradient flow equation is given as

$$\begin{aligned}\frac{\partial \boldsymbol{\theta}_\tau}{\partial \tau} &= -\eta \frac{\partial L(\boldsymbol{\theta}_\tau)}{\partial \boldsymbol{\theta}_\tau} \\ &= -\frac{\eta}{N} \sum_{i=1}^{N} (g(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau) - y_i) \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau)}{\partial \boldsymbol{\theta}_\tau}. \end{aligned} \quad (4)$$

Considering the formulation of the gradient flow in the function space using Equation (4), we obtain

$$\begin{aligned}&\frac{\partial g(\boldsymbol{x}_j, \boldsymbol{\theta}_\tau)}{\partial \tau} \\ &= -\frac{\eta}{N} \sum_{i=1}^{N} (g(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau) - y_i) \underbrace{\left\langle \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau)}{\partial \boldsymbol{\theta}_\tau}, \frac{\partial g(\boldsymbol{x}_j, \boldsymbol{\theta}_\tau)}{\partial \boldsymbol{\theta}_\tau} \right\rangle}_{\text{Neural Tangent Kernel: } \widehat{\Theta}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j)}. \end{aligned} \quad (5)$$

Here, we can see the NTK $\widehat{\Theta}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j)$. With two input datasets $\boldsymbol{x} \in \mathbb{R}^{F\times N}$ and $\boldsymbol{x}' \in \mathbb{R}^{F\times N'}$, the NTK matrix $\widehat{\boldsymbol{H}}_\tau(\boldsymbol{x}, \boldsymbol{x}')$ with the shape of $\mathbb{R}^{N\times N'}$ is formulated as $[\widehat{\boldsymbol{H}}_\tau(\boldsymbol{x}, \boldsymbol{x}')]_{i,j} = \widehat{\Theta}_\tau(\boldsymbol{x}_i, \boldsymbol{x}'_j)$.

From Equation (5), if the NTK does not change during training, the formulation of the gradient flow in the function space becomes a simple ordinary differential equation, and it becomes possible to analytically calculate how the model's output changes during training. When the NTK is positive definite, it is known that the kernel does not change from its initial value during the gradient descent with an infinitesimal step size when considering an infinite number of soft trees (Lee et al., 2019; Kanoh & Sugiyama, 2022) under the formulation described in Section 2.1.

Although the NTK itself is derived from the training with the squared error loss function, insights gained using the NTK extend to various tasks including classification tasks. For example, it has been shown that the Support Vector Machine (SVM) (Hearst et al., 1998) using the NTK is equivalent to the training of the model which induces the same NTK with a soft margin loss function (Chen et al., 2021b).

The NTK induced by a typical soft tree ensemble with infinitely many trees is known to be obtained in closed-form at initialization.

Theorem 2.1 (Kanoh & Sugiyama (2023))**.** *Assume that all M trees have the same tree architecture. Let $\mathcal{Q} : \mathbb{N} \to \mathbb{N} \cup \{0\}$ be a function that takes the depth as input and returns the number of leaves connected to internal nodes at that depth. For any given tree architecture, as the number of trees M goes to infinity, the NTK probabilistically converges to the following deterministic limiting kernel:*

$$\begin{aligned}&\Theta^{\text{Oblique}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &:= \lim_{M\to\infty} \widehat{\Theta}_0^{\text{Oblique}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= \sum_{d=1}^{D} \mathcal{Q}(d) \left(d\, \Sigma_{\{i,j\}} \mathcal{T}_{\{i,j\}}^{d-1} \dot{\mathcal{T}}_{\{i,j\}} + \mathcal{T}_{\{i,j\}}^{d} \right), \end{aligned} \quad (6)$$

where $\mathcal{T}_{\{i,j\}} = \mathbb{E}[\sigma(\boldsymbol{u}^\top \boldsymbol{x}_i + \beta v)\sigma(\boldsymbol{u}^\top \boldsymbol{x}_j + \beta v)]$, $\dot{\mathcal{T}}_{\{i,j\}} = \mathbb{E}[\dot{\sigma}(\boldsymbol{u}^\top \boldsymbol{x}_i + \beta v)\dot{\sigma}(\boldsymbol{u}^\top \boldsymbol{x}_j + \beta v)]$, and $\Sigma_{\{i,j\}} = \boldsymbol{x}_i^\top \boldsymbol{x}_j + \beta^2$. Here, the values of the vector $\boldsymbol{u} \in \mathbb{R}^F$ and the scalar

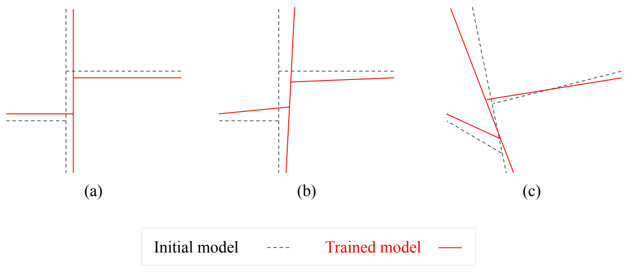

Figure 1. Splitting boundaries. (a): AAA, Always Axis-Aligned, (b): AAI, Axis-Aligned at Initialization, but not during training, (c): Oblique splitting conducted by typical soft trees.

$v \in \mathbb{R}$ are sampled from zero-mean i.i.d. Gaussians with unit variance. Furthermore, if the decision function is the scaled error function, by denoting $\boldsymbol{x}_i^\top \boldsymbol{x}_j + \beta^2$ as $A_{\{i,j\}}$, $\mathcal{T}_{\{i,j\},s}$ and $\dot{\mathcal{T}}_{\{i,j\},s}$ are obtained in closed-form as $\mathcal{T}_{\{i,j\}} = \frac{1}{2\pi}\arcsin\left(\frac{\alpha^2 A_{\{i,j\}}}{\sqrt{(\alpha^2 A_{\{i,i\}}+0.5)(\alpha^2 A_{\{j,j\}}+0.5)}}\right) + \frac{1}{4}$, $\dot{\mathcal{T}}_{\{i,j\}} = \frac{\alpha^2}{\pi}\frac{1}{\sqrt{(1+2\alpha^2 A_{\{i,i\}})(1+2\alpha^2 A_{\{j,j\}})-4\alpha^4 A_{\{i,j\}}^2}}$.

This kernel has rotational invariance with respect to input data.

3. The Theory of the NTK Induced by Axis-Aligned Trees

We first formulate the axis-aligned splitting (Section 3.1), and consider the NTK induced by axis-aligned soft tree ensembles, composed of weak learners with identical architectures, as assumed in Theorem 2.1 (Section 3.2). We then extend it to handle different tree architectures simultaneously (Section 3.3). Detailed proofs can be found in Appendices A and B.

3.1. Setup on Axis-Aligned Splitting

In Equation (1), input to the decision function σ includes the inner product of F-dimensional vectors $\boldsymbol{w}_{m,n}$ and $\boldsymbol{x}_i$. Since $\boldsymbol{w}_{m,n}$ is typically initialized randomly, which is also assumed in Theorem 2.1, the splitting is generally oblique. Thus, Theorem 2.1 cannot directly treat axis-aligned tree ensembles.

To overcome this issue, we analyze the NTK when all the elements except one are set to be zero for every randomly initialized vector $\boldsymbol{w}_{m,n}$. This setting means that the corresponding features are eliminated from consideration of the splitting direction. This is technically not straightforward, as Gaussian random initialization is generally assumed in the existing NTK approaches. Parameters $\boldsymbol{b}$ and $\boldsymbol{\pi}$ are initialized with random values, as is common practice.

We conduct a theoretical analysis of two cases: one where the parameters with zero initialization are not updated during training, as illustrated in Figure 1(a), and the other where they are updated during training in Figure 1(b). These two cases are referred to as AAA (“A”lways “A”xis-“A”ligned) and AAI (“A”xis-“A”ligned at “I”nitialization, but not during training) in this paper.

The learning process of decision trees can be divided into the two following components:

1. Selecting tree topological structure and splitting features.
2. Tuning parameters for splitting thresholds at internal nodes and leaves.

Our NTK-based theoretical analysis of AAA and AAI in the next subsection accounts for the second component. The place of the non-zero element of $\boldsymbol{w}_{m,n}$, which corresponds to the feature assigned to a node in a tree, needs to be predetermined before tuning parameters. Since it can be combined with any method for the first component, the entire model is no longer restrictive in practice. For example, similar to the typical decision trees, it is possible to grow a tree by trying out several splitting patterns and adopting the ones with better performance. In Section 4.2, we address the first component by combining multiple kernels for multiple tree architectures via MKL.

3.2. The NTK Induced by Axis-Aligned Soft Trees

For an input vector $\boldsymbol{x}_i$, let $x_{i,s} \in \mathbb{R}$ be the s-th component of $\boldsymbol{x}_i$. For both AAA and AAI conditions, at initialization, we derive the NTK induced by axis-aligned soft tree ensembles in a closed-form as the number of trees $M \to \infty$.

Theorem 3.1. *Assume that all M trees have the same tree architecture. Let $\{a_1, \ldots, a_\ell, \ldots, a_\mathcal{L}\}$ denote the set of decomposed paths of the trees from the root to the leaves, and let $h(a_\ell) \subset \mathbb{N}$ be the set of feature indices used in splits of the input path a_ℓ. For any tree architecture, as the number of trees M goes to infinity, the NTK probabilistically converges to the following deterministic limiting kernel:*

$$\begin{aligned}&\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\\ :=& \lim_{M\to\infty} \widehat{\Theta}_0^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\\ =& \underbrace{\sum_{\ell=1}^{\mathcal{L}} \sum_{s\in h(a_\ell)} \left(\Sigma_{\{i,j\},s} \dot{\mathcal{T}}_{\{i,j\},s} \prod_{t\in h(a_\ell)\setminus\{s\}} \mathcal{T}_{\{i,j\},t} \right)}_{\text{contribution from internal nodes}}\\ &+ \underbrace{\sum_{\ell=1}^{\mathcal{L}} \prod_{u\in h(a_\ell)} \mathcal{T}_{\{i,j\},u}}_{\text{contribution from leaves}}, \qquad (7)\end{aligned}$$

where $\mathcal{T}_{\{i,j\},s} = \mathbb{E}[\sigma(ux_{i,s} + \beta v)\sigma(ux_{j,s} + \beta v)]$ and $\dot{\mathcal{T}}_{\{i,j\},s} = \mathbb{E}[\dot{\sigma}(ux_{i,s} + \beta v)\dot{\sigma}(ux_{j,s} + \beta v)]$. Here, scalars

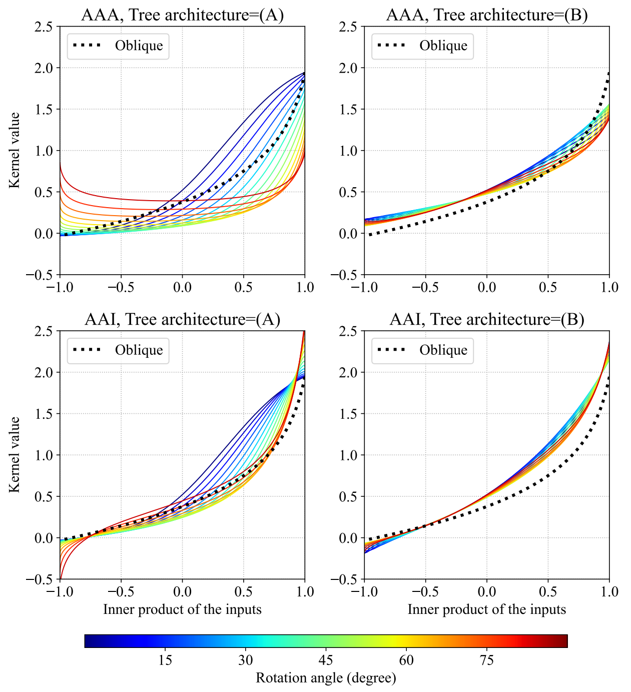

Figure 2. The rotation angle dependency of $\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $\alpha = 2.0$ and $\beta = 0.5$. Different training procedures, AAA and AAI, are listed vertically, and two settings of tree architectures are listed horizontally. The dotted lines show the limiting NTK induced by typical oblique soft tree ensembles defined in Theorem 2.1, which is rotational invariant.

$u, v \in \mathbb{R}$ are sampled from zero-mean i.i.d. Gaussians with unit variance. For $\Sigma_{\{i,j\},s}$, it is $x_{i,s}x_{j,s} + \beta^2$ when AAA is used, and $\boldsymbol{x}_i^\top \boldsymbol{x}_j + \beta^2$ when AAI is used. Furthermore, if the decision function is the scaled error function, by denoting $x_{i,s}x_{j,s} + \beta^2$ as $A_{\{i,j\},s}$, $\mathcal{T}_{\{i,j\},s}$ and $\dot{\mathcal{T}}_{\{i,j\},s}$ are obtained in closed-form as

$$\mathcal{T}_{\{i,j\},s} = \frac{1}{2\pi}\arcsin\left(\frac{\alpha^2 A_{\{i,j\},s}}{\sqrt{(\alpha^2 A_{\{i,i\},s}+0.5)(\alpha^2 A_{\{j,j\},s}+0.5)}}\right)+\frac{1}{4},$$

$$\dot{\mathcal{T}}_{\{i,j\},s} = \frac{\alpha^2}{\pi}\frac{1}{\sqrt{(1+2\alpha^2 A_{\{i,i\},s})(1+2\alpha^2 A_{\{j,j\},s})-4\alpha^4 A^2_{\{i,j\},s}}}.$$

Since we are considering axis-aligned constraints, only a single feature is used at each split for every input path a_ℓ. It is straightforward to extend this formulation and allow multiple features at each split. In an extreme case, if all features are always used at every split, this formula matches the formulation for arbitrary soft trees without axis-aligned constraints in Theorem 2.1.

The difference between AAA and AAI is whether partial features of inputs are used or all features are used in $\Sigma_{\{i,j\},s}$. In AAA, the impact of features that are not used for splitting is completely ignored, while in AAI, the kernel is affected by all features through the inner product of the inputs.

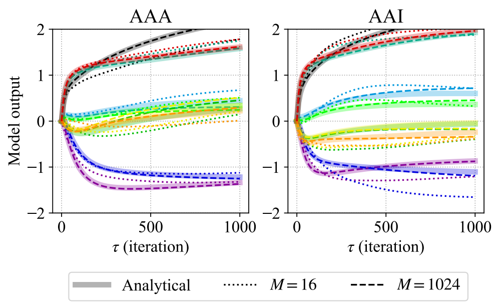

Figure 3. Output dynamics of test data points for axis-aligned soft tree ensembles with two conditions. (Left): AAA, (Right): AAI. Each data point is represented by a different line color. Both sides of the figure are created using exactly the same training and test data.

Figure 2 shows $\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. We set $\alpha = 2.0$ and $\beta = 0.5$. We calculated the kernel values for two rotated vectors: $\boldsymbol{x}_i = (\cos(\omega), \sin(\omega))$, $\boldsymbol{x}_j = (\cos(\omega+\phi), \sin(\omega+\phi))$ where $\omega \in [0, \pi/2]$ and $\phi \in [0, \pi]$. The line colors show ω, and the x-axis shows ϕ. We use a perfect binary tree with depth 2, where we use the first feature at both depths 1 and 2 for the architecture (A) (left column), and we use the first feature at depth 1 and the second feature at depth 2 for (B) (right column). We can see that rotational invariance for the input has disappeared, which is different from the NTK induced by typical soft tree ensembles, shown by the dotted lines (Theorem 2.1). Moreover, when we compare the left and right plots, we can see that the kernel varies depending on the features used for splitting. Appendices E.1 and E.2 show how the NTK induced by a finite number of trees converges to the limiting NTK as the number of trees increases, and the visualization of the kernel when changing hyperparameters.

Figure 3 shows that, for both AAA and AAI, as the number of trees increases, the trajectory obtained analytically from the limiting kernel and the trajectory during gradient descent training become more similar. This result validates the use of the NTK framework for analyzing training behavior. For our experiment, we consider an ensemble of perfect binary trees with $\alpha = 2.0, \beta = 0.5$, where the first feature is used for splitting at depth 1 and the second feature at depth 2. The training and test datasets contain 10 randomly generated $F = 2$ dimensional points each. The prediction targets are also randomly generated. The models with $M = 16$ and 1024 are trained using full-batch gradient descent with a learning rate of 0.1. The initial outputs are shifted to zero (Chizat et al., 2019). Based on Lee et al. (2019), to derive analytical trajectories, we use the limiting kernel (Theorem 3.1), as $f(\boldsymbol{\nu}, \boldsymbol{\theta}_\tau) = \boldsymbol{H}(\boldsymbol{\nu}, \boldsymbol{x})\boldsymbol{H}(\boldsymbol{x}, \boldsymbol{x})^{-1}(\boldsymbol{I} - \exp[-\eta \boldsymbol{H}(\boldsymbol{x}, \boldsymbol{x})\tau])\boldsymbol{y}$, where $\boldsymbol{H}$ is a function that returns the limiting NTK matrix for two input matrices, and $\boldsymbol{I}$ represent an identity matrix. The input

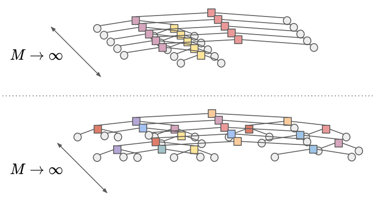

Figure 4. Ensemble of trees with different tree architectures. The color of tree nodes indicates a feature used for splitting.

vector $\boldsymbol{\nu} \in \mathbb{R}^{F\times 1}$ is arbitrary, and the training dataset and targets are denoted by $\boldsymbol{x} \in \mathbb{R}^{F\times N}$ and $\boldsymbol{y} \in \mathbb{R}^{N}$, respectively. The behavior of the prediction trajectory changes depending on the configurations (AAA or AAI), even when exactly the same training and test data are used. In Appendix E.3, we present results from a real-world dataset, where a similar trend can be seen.

3.3. The NTK Induced by Ensembles of Various Trees

In Section 3.2, we have followed the prior work (as per Theorem 2.1) and assumed that soft tree ensembles consist of weak learners with identical architectures, as shown on the top side of Figure 4. However, it is more practical if tree structures and features for splitting vary within an ensemble, as illustrated on the bottom side of Figure 4. To address this issue, we theoretically analyze ensembles with various tree architectures mixed together. Assuming the existence of an infinite number of trees for each architecture in an ensemble, the NTK can be computed analytically using the amount (ratio) of each architecture in the ensemble.

Proposition 3.1. *For any input $\boldsymbol{x}_i$, let $p(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau)$ be the sum of two model functions $q(\boldsymbol{x}_i, \boldsymbol{\theta}'_\tau)$ and $r(\boldsymbol{x}_i, \boldsymbol{\theta}''_\tau)$, where $\boldsymbol{\theta}'_\tau \in \mathbb{R}^{P'}$ and $\boldsymbol{\theta}''_\tau \in \mathbb{R}^{P''}$ are trainable parameters and $\boldsymbol{\theta}_\tau$ is the concatenation of $\boldsymbol{\theta}'_\tau$ and $\boldsymbol{\theta}''_\tau$ used as trainable parameters of p. For any input pair $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the NTK induced by p is equal to the sum of the NTKs of q and r: $\widehat{\Theta}^{(p)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) = \widehat{\Theta}^{(q)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) + \widehat{\Theta}^{(r)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j)$.*

For example, let q and r be functions that represent perfect binary tree ensemble models with a depth of 1 and 2, respectively. In this case, the NTK induced by mixed trees with depths of 1 and 2 is the sum of the NTK induced by trees with a depth of 1 and the NTK induced by trees with a depth of 2. Note that one can straightforwardly generalize it to ensembles containing various tree architectures.

Proposition 3.1 is particularly relevant in the context of axis-aligned trees, as it is impractical to have identical features for splitting across all trees. In addition, this proposition is applicable not only to tree ensembles, but also to various other models. For example, the Neural Additive Model (NAM) (Agarwal et al., 2021), which is a GAM using neural networks, can be treated using this proposition.

4. Insights Derived from the NTK Induced by Axis-Aligned Trees

We further investigate the properties of the NTK induced by the axis-aligned tree ensembles based on our theoretical analysis (Section 4.1) and real-world datasets (Section 4.2) to elucidate the insights of our contribution.

4.1. Sufficiency of the Oblivious Tree for Architecture Candidates

The oblivious tree architecture is a practical perfect binary tree design in which the decision rules for tree splitting are shared across the same depth. This approach reduces the number of required splitting calculations from an exponential time and space complexity of $\mathcal{O}(2^D)$ to a linear complexity of $\mathcal{O}(D)$, where D represents the depth of the perfect binary tree. This property makes the oblivious tree architecture a popular choice in open-source libraries such as CatBoost (Prokhorenkova et al., 2018) and NODE (Popov et al., 2020). Kanoh & Sugiyama (2022) demonstrated that parameter sharing used in oblivious trees does not affect the NTK of soft tree ensembles. However, their analysis does not offer any insights in the context of axis-aligned setups. In addition, they only show the equivalence between perfect binary trees without parameter sharing and their corresponding oblivious trees with the same topological structure.

With Theorem 3.1 and Proposition 3.1, we show that we can always convert axis-aligned non-oblivious tree ensembles, including those with non-perfect binary tree architectures, into axis-aligned oblivious tree ensembles that induce exactly the same limiting NTK.

Proposition 4.1. *For any ensemble of infinitely many axis-aligned trees with the same architecture, one can always construct a set of ensembles of axis-aligned oblivious trees that induce the same limiting NTK, up to constant multiples.*

Detailed proof is provided in Appendix C.

This proposition means that there is no need to consider combinations of complex trees, and it is sufficient to consider only combinations of oblivious trees. Although various trial-and-error processes are necessary for model selection to determine features used at each node, this finding can reduce the number of processes by excluding non-oblivious trees from the search space, leading to the potential for faster model architecture determination. Although the NTK values differ by a constant factor, this does not, theoretically, have a significant impact. As Equation (5) shows, even if two kernels are different only up to a constant, adjusting the learning rate η can make their training behavior the same.

Figure 5 shows the empirical validation of Proposition 4.1. We consider an asymmetric non-oblivious tree architecture, where the first feature is used for splitting at depth 1 and

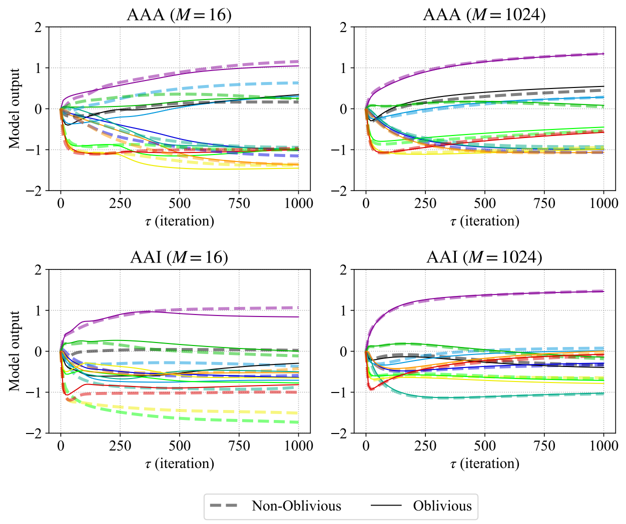

Figure 5. Output dynamics of test data points for axis-aligned soft tree ensembles under four conditions. (Top left): AAA with $M = 16$, (Top right): AAA with $M = 1024$, (Bottom left): AAI with $M = 16$, (Bottom right): AAI with $M = 1024$. Dashed and solid lines represent the asymmetric tree model and the oblivious trees converted using Proposition 4.1, respectively. Each data point is represented by a different line color. All plots are created using exactly the same training and test data.

the second feature is used for splitting at depth 2. The left child of the first splitting node is not a splitting node but a leaf node. For this architecture, with Proposition 4.1, combining two oblivious tree architectures induces the same limiting NTK. Detailed conversion procedure is provided in Appendix C. Using Figure 5, for $M = 16$ and $M = 1024$, we can verify whether the trajectories trained under the conditions of AAA and AAI match those obtained using the converted oblivious trees using Proposition 4.1. The results show that if there are a total of 1024 trees, the behavior before and after the conversion is consistent. Note that in the case of oblivious trees, since there are two tree architectures after conversion, each tree architecture has 8 and 512 trees, respectively, so that the total number of trees is 16 and 1024, respectively. The method for training a finite number of trees and the process for generating datasets are the same as those in Figure 3.

4.2. Empirical Investigation via MKL

We examine the behavior of the NTK on real-world datasets in terms of feature selection and generalization performance. Our theoretical analysis in Section 3 assumes that features used at nodes are predetermined. To alleviate this limitation and include feature selection, we use MKL (Gönen & Alpaydın, 2011), which determines the weights of a linear combination of multiple kernels via training. Using NTKs induced by various tree architectures in MKL, we can learn how much each tree architecture should contribute (Proposition 3.1), which can be also interpreted as Neural Architecture Search (NAS) (Elsken et al., 2019; Chen et al., 2021a; Xu et al., 2021; Mok et al., 2022).

We use EasyMKL (Aiolli & Donini, 2015), a convex approach that identifies kernel combinations maximizing the margin between classes. Figure 6 displays the weights obtained by EasyMKL on the entire tic-tac-toe dataset preprocessed by Fernández-Delgado et al. (2014). Tic-tac-toe is a two-player game in which the objective is to form a line of three consecutive symbols (either "X" or "O") horizontally, vertically, or diagonally on a 3×3 grid. The tic-tac-toe dataset consists of $F = 3 \times 3 = 9$ features, each of which indicates the status, "X", "O", or blank, of the corresponding position on the game board, and the task is a classification of the outcome based on this data. We enumerate all the combination patterns from the first to the third order and use EasyMKL to determine the linear combination of $\binom{F}{1} + \binom{F}{2} + \binom{F}{3} = 129$ kernels with $\alpha = 2.0$ and $\beta = 0.5$. The number of potential tree architectures exceeds 129 when considering the order of splits within the tree. However, as indicated in Proposition 4.1, it is sufficient to focus on only the oblivious tree, which allows us to ignore the order of splits. Hence, those 129 patterns encompass all the valid configurations, which is one of the advantages of our approach.

As shown in Figure 6, for AAA, interactions that are essential to determine the outcome of the game carry significant weights. In contrast, for AAI, weights tend to be more uniform. This suggests that AAA is more sensitive to the nature of data than AAI, while a simple approach that randomly selects diverse tree architectures can be effective for AAI in practice. Such a trend appears to hold true not only for the tic-tac-toe dataset, but across a wide range of datasets. Details can be found in Appendix E.5.

We also analyze the generalization performance on the tic-tac-toe dataset. Three types of the limiting NTKs induced by the soft tree ensembles are employed: AAA, AAI (Theorem 3.1) and Oblique (Theorem 2.1), as shown in Figure 1. For the oblique kernel, we assumed a perfect binary tree structure and, since AAA and AAI consider interactions up to the third order, we set the tree depth to 3. The SVM with these kernels was used for classification. Kernel parameters were set with α in $\{0.5, 1.0, 2.0, 4.0\}$ and β in $\{0.1, 0.5, 1.0\}$. We used the regularization strength $C = 1.0$ in SVMs. For both AAA and AAI, a total of $\binom{F}{1} + \binom{F}{2} + \binom{F}{3} = 129$ kernels were prepared and three types of weights for the linear combination of these kernels were tested. The weight of the first type, called "MKL", is obtained by EasyMKL. The second, called "Optimal", is $1/8$ if the interaction determines the outcome of the tic-tac-toe game (there are eight such interactions) and 0 otherwise.

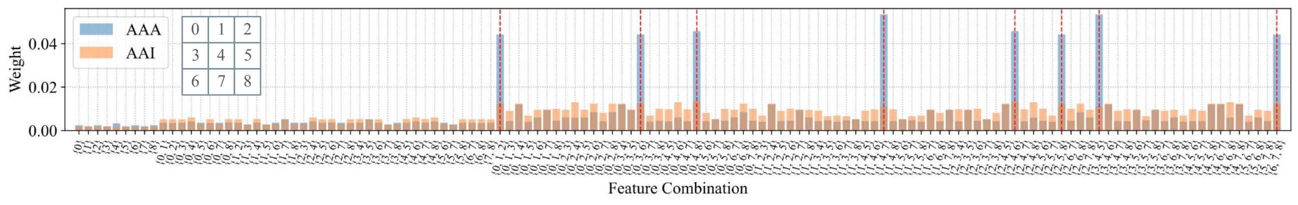

Figure 6. Weights of a linear combination of multiple kernels obtained using EasyMKL. The interactions highlighted by red dotted vertical lines indicate the feature combinations that determine the outcome of the tic-tac-toe game. The correspondence between the game board and the feature indices is displayed on the left side of the figure.

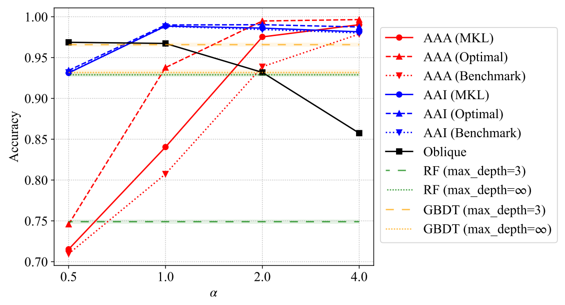

Figure 7. Classification accuracy on the tic-tac-toe dataset. The performance of RF/GBDT does not depend on α, and is represented by the horizontal line. Since RF/GBDT have randomness, its standard deviation of the four-fold cross-validation accuracy in 12 executions is shown by a semi-transparent band.

The third type of weight, called "Benchmark", is uniform for all kernels. Additionally, as reference information, we present the performance of Random Forest (RF) and Gradient Boosting Decision Trees (GBDT).

Figure 7 displays the results of four-fold cross-validation, where 25 percent of the total amount of data were used for training and the remainder for evaluation. No significant variations were observed when adjusting β, so we present results with $\beta = 0.5$. For RF/GBDT, the number of weak learners is set to 1000. Detailed experimental results, including those obtained by modifying β and comparisons with forest models under diverse configurations, are provided in Appendices E.4 and E.6. It is evident from the results that setting appropriate weights for each interaction improves generalization performance. This improvement is particularly remarkable in AAA; that is, AAA (MKL) and AAA (Optimal) are superior to AAA (Benchmark) across all α. For AAI, the performance is comparable between AAI (Optimal) and AAI (Benchmark), which is consistent with the insights obtained from Figure 6.

Since axis-aligned models and gradient methods have their own inductive biases, using them could improve generalization performance in different ways. By considering the tic-tac-toe dataset as a case study, we explore the benefit of training axis-aligned models using gradient methods.

Axis-Aligned / Oblique. Under the optimal hyperparameters, the performance was ranked in the order of AAA, AAI, and then Oblique. Thus a stronger inductive bias leads to more accurate results on this dataset. This emphasizes the relevance of axis-aligned models in practical situations.

Gradient Descent / Greedy Search. The comparison between AAA and RF/GBDT suggests that gradient descent-based learning even without MKL, referred to as "Benchmark", is more effective than a greedy approach for this dataset. The outcome of the tic-tac-toe game, the output variable of this dataset, is determined only by the third-order interactions of features, and it seems that greedy approaches are not appropriate to pick up such interactions.

This example highlights the importance of considering axis-aligned models trained by gradient descent, as treated in our study and literature (Popov et al., 2020; Chang et al., 2022). Although discussions about generalization performance are always data-dependent and do not immediately apply to other datasets, axis-aligned models trained with a gradient-based method perform well not only on the tic-tac-toe dataset, but also on several datasets as described in Appendix E.6. Our insights, such as the sufficiency of oblivious trees in tree architecture search (Section 4.1) and the tight connection between feature selection and the type of constraints on training (Section 4.2), can serve as a basis of further development of large tree models.

5. Conclusion

In this paper, we have formulated the NTK induced by the axis-aligned soft tree ensembles, and have succeeded in describing the analytical training trajectory. We have theoretically analyzed two scenarios: one where the axis-aligned constraint is applied throughout the training process, and the other where the initial model is axis-aligned and training proceeds without any constraints. We have also presented a theoretical framework to deal with non-identical tree architectures simultaneously and used it to provide theoretical support for the validity of using oblivious trees. Furthermore, using MKL, we have shown that the suitable features for AAA and AAI can be different from each other. Our work highlights how tree architecture and constraints affect the model behavior and provides insights into the design of tree-based models.

Acknowledgements

This work was supported by JSPS, KAKENHI Grant Number JP21H03503, Japan and JST, CREST Grant Number JPMJCR22D3, Japan.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agarwal, R., Melnick, L., Frosst, N., Zhang, X., Lengerich, B., Caruana, R., and Hinton, G. Neural Additive Models: Interpretable Machine Learning with Neural Nets. In *Advances in Neural Information Processing Systems*, 2021.

Aiolli, F. and Donini, M. EasyMKL: a scalable multiple kernel learning algorithm. *Neurocomputing*, 2015.

Aiolli, F., Da San Martino, G., and Sperduti, A. A Kernel Method for the Optimization of the Margin Distribution. In *Artificial Neural Networks*, 2008.

Arik, S. Ã. and Pfister, T. TabNet: Attentive Interpretable Tabular Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Arora, S., Du, S. S., Li, Z., Salakhutdinov, R., Wang, R., and Yu, D. Harnessing the Power of Infinitely Wide Deep Nets on Small-data Tasks. In *International Conference on Learning Representations*, 2020.

Chang, C.-H., Caruana, R., and Goldenberg, A. NODE-GAM: Neural generalized additive model for interpretable deep learning. In *International Conference on Learning Representations*, 2022.

Chen, W., Gong, X., and Wang, Z. Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective. In *International Conference on Learning Representations*, 2021a.

Chen, Y., Huang, W., Nguyen, L., and Weng, T.-W. On the Equivalence between Neural Network and Support Vector Machine. In *Advances in Neural Information Processing Systems*, 2021b.

Chizat, L., Oyallon, E., and Bach, F. On Lazy Training in Differentiable Programming. In *Advances in Neural Information Processing Systems*, 2019.

Elsken, T., Metzen, J. H., and Hutter, F. Neural Architecture Search: A Survey. *Journal of Machine Learning Research*, 2019.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research*, 2014.

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. Sharpness-aware Minimization for Efficiently Improving Generalization. In *International Conference on Learning Representations*, 2021.

Frosst, N. and Hinton, G. E. Distilling a Neural Network Into a Soft Decision Tree. *CoRR*, 2017.

Gönen, M. and Alpaydın, E. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12, 2011.

Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Hastie, T. and Tibshirani, R. Generalized Additive Models. *Statistical Science*, 1986.

Hazimeh, H., Ponomareva, N., Mol, P., Tan, Z., and Mazumder, R. The Tree Ensemble Layer: Differentiability meets Conditional Computation. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Hearst, M., Dumais, S., Osuna, E., Platt, J., and Scholkopf, B. Support vector machines. *IEEE Intelligent Systems and their Applications*, 1998.

Humbird, K. D., Peterson, J. L., and Mcclarren, R. G. Deep Neural Network Initialization With Decision Trees. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

Jacot, A., Gabriel, F., and Hongler, C. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. In *Advances in Neural Information Processing Systems*, 2018.

Joseph, M. PyTorch Tabular: A Framework for Deep Learning with Tabular Data. *CoRR*, 2021.

Kanoh, R. and Sugiyama, M. A Neural Tangent Kernel Perspective of Infinite Tree Ensembles. In *International Conference on Learning Representations*, 2022.

Kanoh, R. and Sugiyama, M. Analyzing Tree Architectures in Ensembles via Neural Tangent Kernel. In *International Conference on Learning Representations*, 2023.

Ke, G., Xu, Z., Zhang, J., Bian, J., and Liu, T.-Y. DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

Kingma, D. and Ba, J. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Kontschieder, P., Fiterau, M., Criminisi, A., and BulÃš, S. R. Deep Neural Decision Forests. In *IEEE International Conference on Computer Vision*, 2015.

Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. In *Advances in Neural Information Processing Systems*, 2019.

Martins, A. and Astudillo, R. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

Mok, J., Na, B., Kim, J.-H., Han, D., and Yoon, S. Demystifying the Neural Tangent Kernel from a Practical Perspective: Can it be trusted for Neural Architecture Search without training? In *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

Peters, B., Niculae, V., and Martins, A. F. T. Sparse Sequence-to-Sequence Models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Popov, S., Morozov, S., and Babenko, A. Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data. In *International Conference on Learning Representations*, 2020.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, 2018.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014.

Xu, J., Zhao, L., Lin, J., Gao, R., Sun, X., and Yang, H. KNAS: Green Neural Architecture Search. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

A. Proof of Theorem 3.1

Theorem 3.1. *Assume that all M trees have the same tree architecture. Let $\{a_1, \ldots, a_\ell, \ldots, a_{\mathcal{L}}\}$ denote the set of decomposed paths of the trees from the root to the leaves, and let $h(a_\ell) \subset \mathbb{N}$ be the set of feature indices used in splits of the input path a_ℓ. For any tree architecture, as the number of trees M goes to infinity, the NTK probabilistically converges to the following deterministic limiting kernel:*

$$\begin{aligned}\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j) &\coloneqq \lim_{M\to\infty} \widehat{\Theta}_0^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\\ &= \sum_{\ell=1}^{\mathcal{L}} \left(\sum_{s\in h(a_\ell)} \Sigma_{\{i,j\},s} \dot{\mathcal{T}}_{\{i,j\},s} \prod_{t\in h(a_\ell)\setminus\{s\}} \mathcal{T}_{\{i,j\},t} + \prod_{s\in h(a_\ell)} \mathcal{T}_{\{i,j\},s} \right),\end{aligned} \tag{A.1}$$

where $\mathcal{T}_{\{i,j\},s} = \mathbb{E}[\sigma(ux_{i,s} + \beta v)\sigma(ux_{j,s} + \beta v)]$ and $\dot{\mathcal{T}}_{\{i,j\},s} = \mathbb{E}[\dot{\sigma}(ux_{i,s} + \beta v)\dot{\sigma}(ux_{j,s} + \beta v)]$. Here, scalars $u, v \in \mathbb{R}$ are sampled from zero-mean i.i.d. Gaussians with unit variance. For $\Sigma_{\{i,j\},s}$, it is $x_{i,s}x_{j,s} + \beta^2$ when AAA is used, and $\boldsymbol{x}_i^\top \boldsymbol{x}_j + \beta^2$ when AAI is used. Furthermore, if the decision function is the scaled error function, $\mathcal{T}_{\{i,j\},s}$ and $\dot{\mathcal{T}}_{\{i,j\},s}$ are obtained in closed-form as

$$\mathcal{T}_{\{i,j\},s} = \frac{1}{2\pi} \arcsin\left(\frac{\alpha^2(x_{i,s}x_{j,s} + \beta^2)}{\sqrt{(\alpha^2(x_{i,s}^2 + \beta^2) + 0.5)(\alpha^2(x_{j,s}^2 + \beta^2) + 0.5)}} \right) + \frac{1}{4}, \tag{A.2}$$

$$\dot{\mathcal{T}}_{\{i,j\},s} = \frac{\alpha^2}{\pi} \frac{1}{\sqrt{\left(1 + 2\alpha^2(x_{i,s}^2 + \beta^2)\right)\left(1 + 2\alpha^2(x_{j,s}^2 + \beta^2)\right) - 4\alpha^4(x_{i,s}x_{j,s} + \beta^2)^2}}. \tag{A.3}$$

Proof. Based on the independence of parameters at each leaf and the symmetry of the decision function, Kanoh & Sugiyama (2023) showed that the NTK induced by arbitrary soft tree ensembles can be decomposed into the sum of the NTKs induced by the rule sets, which are constructed by paths from the tree root to leaves. This property also holds in our formulation (Section 2.1). Therefore, we formulate the NTK induced by rule sets and use it to derive the NTK induced by axis-aligned soft tree ensembles.

For simplicity, we first assume $\beta = 0$ in Equation (1). Let D_ℓ be the depth of a rule set, which is a path from the root to a leaf ℓ. We consider the contribution from internal nodes $\Theta^{(D_\ell,\text{Rule,nodes})}$ and the contribution from leaves $\Theta^{(D_\ell,\text{Rule,leaves})}$ separately, such that

$$\Theta^{(D_\ell,\text{Rule})}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Theta^{(D_\ell,\text{Rule,nodes})}(\boldsymbol{x}_i, \boldsymbol{x}_j) + \Theta^{(D_\ell,\text{Rule,leaves})}(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{A.4}$$

As for internal nodes, when we treat the axis-aligned case (Section 3.1), only a single parameter in $\boldsymbol{w}_{m,n}$ is non-zero at initialization. When calculating the NTK as shown in Equation (5), the parameter derivatives in terms of trainable parameters are considered. In the cases of AAA and AAI, they are given as follows at initialization:

$$\frac{\partial f^{(D_\ell,\text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{\pi})}{\partial w_{m,n,k_n}} = \frac{1}{\sqrt{M}} x_{i,k_n} \dot{\sigma}(w_{m,n,k_n} x_{i,k_n}) f_m^{(D_\ell-1,\text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}_{m,-n}, \boldsymbol{\pi}_m), \qquad \text{(AAA)} \tag{A.5}$$

$$\frac{\partial f^{(D_\ell,\text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{\pi})}{\partial \boldsymbol{w}_{m,n}} = \frac{1}{\sqrt{M}} \boldsymbol{x}_i \dot{\sigma}(w_{m,n,k_n} x_{i,k_n}) f_m^{(D_\ell-1,\text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}_{m,-n}, \boldsymbol{\pi}_m), \qquad \text{(AAI)} \tag{A.6}$$

where x_{i,k_n} and w_{m,n,k_n} are k_n-th feature in $\boldsymbol{x}_i$ and k_n-th parameter in $\boldsymbol{w}_{m,n}$, respectively, and $\boldsymbol{w}_{m,-n}$ denotes the internal node parameter matrix except for the parameters of the node n.

As a preliminary step for calculating the NTK, we obtain the following equation at initialization:

$$\begin{aligned}&\mathbb{E}_m \left[f_m^{(D_\ell,\text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{\pi}_m)\, f_m^{(D_\ell,\text{Rule})}(\boldsymbol{x}_j, \boldsymbol{w}_m, \boldsymbol{\pi}_m) \right]\\ &= \mathbb{E}_m \left[\sigma(\boldsymbol{w}_{m,1}^\top \boldsymbol{x}_i)\sigma(\boldsymbol{w}_{m,1}^\top \boldsymbol{x}_j) \cdots \sigma(\boldsymbol{w}_{m,D}^\top \boldsymbol{x}_i)\sigma(\boldsymbol{w}_{m,D}^\top \boldsymbol{x}_j) \pi_{m,\ell}^2 \right]\\ &= \mathbb{E}_m \left[\underbrace{\sigma(w_{m,1,k_1} x_{i,k_1})\sigma(w_{m,1,k_1} x_{j,k_1})}_{\to \mathcal{T}_{\{i,j\},k_1}} \cdots \underbrace{\sigma(w_{m,D_\ell,k_{D_\ell}} x_{i,k_{D_\ell}})\sigma(w_{m,D_\ell,k_{D_\ell}} x_{j,k_{D_\ell}})}_{\to \mathcal{T}_{\{i,j\},k_{D_\ell}}} \underbrace{\pi_{m,\ell}^2}_{\to 1} \right]\end{aligned}$$

$$= \prod_{t \in h(a_\ell)} \mathcal{T}_{\{i,j\},t}, \tag{A.7}$$

where the symbol "$\rightarrow$" denotes the expected value of the corresponding term will take. The transition from the second line to the third line in Equation (A.7) uses the equality $\sigma(\boldsymbol{w}_{m,n}^\top \boldsymbol{x}_i) = \sigma(w_{m,n,k_n} x_{i,k_n})$ at initialization. When the decision function is the scaled error function, Equations (A.2) and (A.3) are obtained using the conventional formulas introduced in Kanoh & Sugiyama (2022), which are also applied in Theorem 2.1.

Using Equation (A.7), the limiting NTK contribution from the n-th node is

$$\begin{aligned}
&\lim_{M\to\infty} \frac{1}{M} \sum_{m=1}^{M} \Bigg(\Sigma_{\{i,j\},k_n} \times \dot{\sigma}(w_{m,n,k_n} x_{i,k_n}) \dot{\sigma}(w_{m,n,k_n} x_{j,k_n}) \\
&\qquad \times f_m^{(D_\ell - 1, \text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{\pi}_m)\, f_m^{(D_\ell - 1, \text{Rule})}(\boldsymbol{x}_j, \boldsymbol{w}_m, \boldsymbol{\pi}_m) \Bigg) \\
&= \Sigma_{\{i,j\},k_n} \times \mathbb{E}_m \Bigg[\underbrace{\dot{\sigma}(w_{m,n,k_n} x_{i,k_n}) \dot{\sigma}(w_{m,n,k_n} x_{j,k_n})}_{\rightarrow \dot{\mathcal{T}}_{\{i,j\},k_n}} \Bigg] \\
&\qquad \times \mathbb{E}_m \Bigg[\underbrace{f_m^{(D_\ell - 1, \text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{\pi}_m)\, f_m^{(D_\ell - 1, \text{Rule})}(\boldsymbol{x}_j, \boldsymbol{w}_m, \boldsymbol{\pi}_m)}_{\rightarrow \prod_{t \in h(a_\ell)\setminus\{k_n\}} \mathcal{T}_{\{i,j\},t}} \Bigg],
\end{aligned} \tag{A.8}$$

where $\Sigma_{\{i,j\},k_n} = x_{i,k_n} x_{j,k_n}$ when AAA is used, and $\Sigma_{\{i,j\},k_n} = \boldsymbol{x}_i^\top \boldsymbol{x}_j$ when AAI is used. Since there are D_ℓ possible locations for n, we obtain

$$\Theta^{(D_\ell, \text{Rule,nodes})}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{s \in h(a_\ell)} \left(\Sigma_{\{i,j\},s} \dot{\mathcal{T}}_{\{i,j\},s} \prod_{t \in h(a_\ell)\setminus\{s\}} \mathcal{T}_{\{i,j\},t} \right). \tag{A.9}$$

For leaves, since

$$\frac{\partial f^{(D_\ell, \text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{\pi})}{\partial \pi_{m,\ell}} = \frac{1}{\pi_{m,\ell}\sqrt{M}} f_m^{(D_\ell, \text{Rule})}(\boldsymbol{x}_i, \boldsymbol{w}_m, \boldsymbol{\pi}_m), \tag{A.10}$$

with Equation (A.7), we have

$$\Theta^{(D_\ell, \text{Rule,leaves})}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \prod_{s \in h(a_\ell)} \mathcal{T}_{\{i,j\},s}. \tag{A.11}$$

Combining Equation (A.9) and Equation (A.11), we obtain

$$\Theta^{(D_\ell, \text{Rule})}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{s \in h(a_\ell)} \left(\Sigma_{\{i,j\},s} \dot{\mathcal{T}}_{\{i,j\},s} \prod_{t \in h(a_\ell)\setminus\{s\}} \mathcal{T}_{\{i,j\},t} \right) + \prod_{s \in h(a_\ell)} \mathcal{T}_{\{i,j\},s}. \tag{A.12}$$

When we sum up this NTK over multiple rule sets constructed by multiple leaves, it becomes the NTK of the axis-aligned soft tree ensembles:

$$\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{\ell=1}^{\mathcal{L}} \left(\sum_{s \in h(a_\ell)} \Sigma_{\{i,j\},s} \dot{\mathcal{T}}_{\{i,j\},s} \prod_{t \in h(a_\ell)\setminus\{s\}} \mathcal{T}_{\{i,j\},t} + \prod_{s \in h(a_\ell)} \mathcal{T}_{\{i,j\},s} \right). \tag{A.13}$$

Up to this point, we have been considering the case of $\beta = 0$. It is straightforward to take the case $\beta \neq 0$ into account because, in the case of soft tree ensemble, the bias term can be represented by using an extra feature that takes a constant value β as input. This allows us to generally express the bias term's contribution by adding β^2 to the section where the product of the inputs is calculated. $\square$

By redefining $h(a_\ell) \subset \mathbb{N}$ as $h(a_\ell) \subset \mathcal{P}(\mathbb{N})$, where $\mathcal{P}$ denotes the power set, and changing the component $x_{i,s} x_{j,s}$ to $\boldsymbol{x}_{i,S}^\top \boldsymbol{x}_{j,S}$, where S is the set of feature indices used in a splitting node, we can handle multiple features at each split.

B. Proof of Proposition 3.1

Proposition 3.1. *For any input $\boldsymbol{x}_i$, let $p(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau)$ be the sum of two model functions $q(\boldsymbol{x}_i, \boldsymbol{\theta}'_\tau)$ and $r(\boldsymbol{x}_i, \boldsymbol{\theta}''_\tau)$, where $\boldsymbol{\theta}'_\tau \in \mathbb{R}^{P'}$ and $\boldsymbol{\theta}''_\tau \in \mathbb{R}^{P''}$ are trainable parameters and $\boldsymbol{\theta}_\tau$ is the concatenation of $\boldsymbol{\theta}'_\tau$ and $\boldsymbol{\theta}''_\tau$ used as trainable parameters of p. For any input pair $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the NTK induced by p is equal to the sum of the NTKs of q and r: $\widehat{\Theta}^{(p)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) = \widehat{\Theta}^{(q)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) + \widehat{\Theta}^{(r)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j)$.*

Proof. The NTK induced by this model can be decomposed into the sum of the NTKs of each tree architecture as follows:

$$\begin{aligned}
\widehat{\Theta}^{(p)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \left\langle \frac{\partial p(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau)}{\partial \boldsymbol{\theta}_\tau}, \frac{\partial p(\boldsymbol{x}_j, \boldsymbol{\theta}_\tau)}{\partial \boldsymbol{\theta}_\tau} \right\rangle \\
&= \left(\tfrac{\partial p(\boldsymbol{x}_i,\boldsymbol{\theta}_\tau)}{\partial \theta'_{\tau,1}} \cdots \tfrac{\partial p(\boldsymbol{x}_i,\boldsymbol{\theta}_\tau)}{\partial \theta'_{\tau,P'}} \, \tfrac{\partial p(\boldsymbol{x}_i,\boldsymbol{\theta}_\tau)}{\partial \theta''_{\tau,1}} \cdots \tfrac{\partial p(\boldsymbol{x}_i,\boldsymbol{\theta}_\tau)}{\partial \theta''_{\tau,P''}} \right) \begin{pmatrix} \frac{\partial p(\boldsymbol{x}_j,\boldsymbol{\theta}_\tau)}{\partial \theta'_{\tau,1}} \\ \vdots \\ \frac{\partial p(\boldsymbol{x}_j,\boldsymbol{\theta}_\tau)}{\partial \theta'_{\tau,P'}} \\ \frac{\partial p(\boldsymbol{x}_j,\boldsymbol{\theta}_\tau)}{\partial \theta''_{\tau,1}} \\ \vdots \\ \frac{\partial p(\boldsymbol{x}_j,\boldsymbol{\theta}_\tau)}{\partial \theta''_{\tau,P''}} \end{pmatrix} \\
&= \left(\tfrac{\partial q(\boldsymbol{x}_i,\boldsymbol{\theta}'_\tau)}{\partial \theta'_{\tau,1}} \cdots \tfrac{\partial q(\boldsymbol{x}_i,\boldsymbol{\theta}'_\tau)}{\partial \theta'_{\tau,P'}} \, \tfrac{\partial r(\boldsymbol{x}_i,\boldsymbol{\theta}''_\tau)}{\partial \theta''_{\tau,1}} \cdots \tfrac{\partial r(\boldsymbol{x}_i\boldsymbol{\theta}''_\tau)}{\partial \theta''_{\tau,P''}} \right) \begin{pmatrix} \frac{\partial q(\boldsymbol{x}_j,\boldsymbol{\theta}'_\tau)}{\partial \theta'_{\tau,1}} \\ \vdots \\ \frac{\partial q(\boldsymbol{x}_j,\boldsymbol{\theta}'_\tau)}{\partial \theta'_{\tau,P'}} \\ \frac{\partial r(\boldsymbol{x}_j,\boldsymbol{\theta}''_\tau)}{\partial \theta''_{\tau,1}} \\ \vdots \\ \frac{\partial r(\boldsymbol{x}_j,\boldsymbol{\theta}''_\tau)}{\partial \theta''_{\tau,P''}} \end{pmatrix} \\
&= \left(\tfrac{\partial q(\boldsymbol{x}_i,\boldsymbol{\theta}'_\tau)}{\partial \theta'_{\tau,1}} \cdots \tfrac{\partial q(\boldsymbol{x}_i,\boldsymbol{\theta}'_\tau)}{\partial \theta'_{\tau,P'}} \right) \begin{pmatrix} \frac{\partial q(\boldsymbol{x}_j,\boldsymbol{\theta}'_\tau)}{\partial \theta'_{\tau,1}} \\ \vdots \\ \frac{\partial q(\boldsymbol{x}_j,\boldsymbol{\theta}'_\tau)}{\partial \theta'_{\tau,P'}} \end{pmatrix} + \left(\tfrac{\partial r(\boldsymbol{x}_i,\boldsymbol{\theta}''_\tau)}{\partial \theta''_{\tau,1}} \cdots \tfrac{\partial r(\boldsymbol{x}_i,\boldsymbol{\theta}''_\tau)}{\partial \theta''_{\tau,P''}} \right) \begin{pmatrix} \frac{\partial r(\boldsymbol{x}_j,\boldsymbol{\theta}''_\tau)}{\partial \theta''_{\tau,1}} \\ \vdots \\ \frac{\partial r(\boldsymbol{x}_j,\boldsymbol{\theta}''_\tau)}{\partial \theta''_{\tau,P''}} \end{pmatrix} \\
&= \underbrace{\left\langle \frac{\partial q(\boldsymbol{x}_i, \boldsymbol{\theta}')}{\partial \boldsymbol{\theta}'}, \frac{\partial q(\boldsymbol{x}_j, \boldsymbol{\theta}')}{\partial \boldsymbol{\theta}'} \right\rangle}_{\widehat{\Theta}^{(q)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j)} + \underbrace{\left\langle \frac{\partial r(\boldsymbol{x}_i, \boldsymbol{\theta}'')}{\partial \boldsymbol{\theta}''}, \frac{\partial r(\boldsymbol{x}_j, \boldsymbol{\theta}'')}{\partial \boldsymbol{\theta}''} \right\rangle}_{\widehat{\Theta}^{(r)}_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j)}.
\end{aligned} \tag{B.1}$$

Here, since q is not a function of $\boldsymbol{\theta}''$ and r is not a function of $\boldsymbol{\theta}'$, the property that their respective derivatives are zero is used in the transition from the second line to the third line in Equation (B.1). □

The NTK decomposition for any number of sub-models follows by repeatedly using this property.

C. Proof of Proposition 4.1

Proposition 4.1. *For any ensemble of infinitely many axis-aligned trees with the same architecture, one can always construct a set of ensembles of axis-aligned oblivious trees that induce the same limiting NTK, up to constant multiples.*

Proof. We prove this proposition using Figure A.1 as an instance. It is straightforward to generalize the following discussion to any trees. As shown in Theorem 3.1, the limiting NTK is characterized by the root-to-leaf paths. Therefore, the limiting NTK induced by an infinite number of trees shown at ① in Figure A.1 is identical to the limiting NTK induced by an infinite number of rule sets shown at ② in Figure A.1. For any root-to-leaf path with the length D_ℓ, one can always construct a single perfect binary tree architecture that induces exactly the same NTK using 2^{D_ℓ} rule sets composed of the same

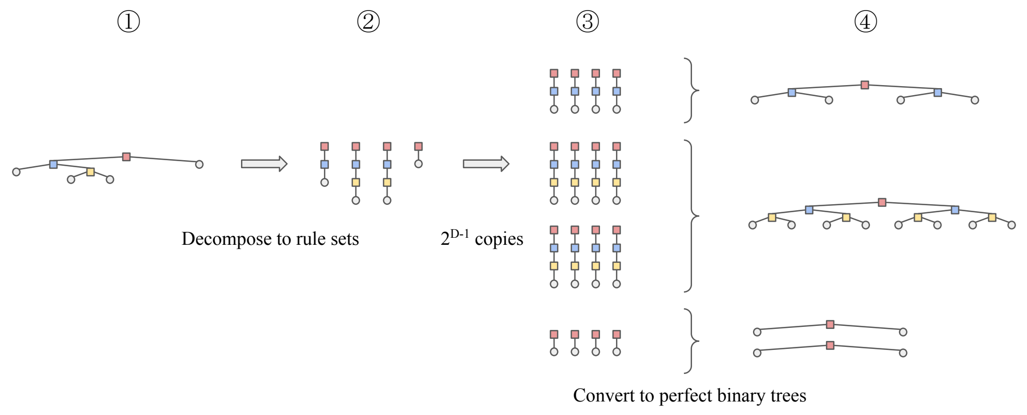

Figure A.1. A procedure to convert any arbitrary binary tree ensemble into a set of perfect binary trees that use the same feature at each depth with the exactly same limiting NTK up to a constant multiple.

features with the path. Since we consider binary splitting at every node, the number of children at each node is 2. Therefore, for the maximum depth D of a tree, by having 2^{D-1} copies of the same tree ensembles (②→③ in Figure A.1), we can convert them into perfect binary trees that use the same feature at each depth (③→④ in Figure A.1). Here, each copy is identical in terms of its graph topological structure and the features used during the initialization of splitting nodes, but their randomly initialized parameters are independent. Note that when there are D copies, the NTK also becomes D times larger. This can be understood using Equation (B.1) by considering the case in Proposition 3.1, where we have $p(\boldsymbol{x}_i, \boldsymbol{\theta}_\tau) = q(\boldsymbol{x}_i, \boldsymbol{\theta}'_\tau) + q(\boldsymbol{x}_i, \boldsymbol{\theta}''_\tau)$.

Until now, even though the same features are used for splitting at each depth, there has been no discussion regarding parameter sharing. Next, we show that axis-aligned oblivious trees and perfect binary trees that do not share parameters shown at ④ induce the same NTK. For simplicity, we first assume $\beta = 0$. For a soft Boolean operation, because of the symmetry of the decision function and independence of parameters at each leaf, even if $1 - \sigma(\boldsymbol{w}_{m,n}^\top \boldsymbol{x}_i)$ is replaced with $\sigma(\boldsymbol{w}_{m,n}^\top \boldsymbol{x}_i)$, the resulting NTK remains unchanged (Kanoh & Sugiyama, 2023). When all $1 - \sigma(\boldsymbol{w}_{m,n}^\top \boldsymbol{x}_i)$ for all n are replaced with $\sigma(\boldsymbol{w}_{m,n}^\top \boldsymbol{x}_i)$, the function converted from the oblivious tree ensemble with depth D becomes:

$$f^{(D,\text{Converted})}(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{\pi}) = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left(\prod_{d=1}^{D} \sigma\left(\boldsymbol{w}_{m,d}^\top \boldsymbol{x}_i\right) \sum_{\ell=1}^{2^D} \pi_{m,\ell} \right). \tag{C.1}$$

We now calculate the NTK for Equation (C.1). For internal nodes at initialization, we obtain

$$\frac{\partial f^{(D,\text{Converted})}(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{\pi})}{\partial w_{m,d,k_d}} = \frac{1}{\sqrt{M}} x_{i,k_d} \dot{\sigma}\left(w_{m,d,k_d} x_{i,k_d}\right) \prod_{s \in \{1,2,3,\ldots,D\} \setminus \{d\}} \sigma\left(w_{m,s,k_s} x_{i,k_s}\right) \sum_{\ell=1}^{2^D} \pi_{m,\ell}, \quad \text{(AAA)} \tag{C.2}$$

$$\frac{\partial f^{(D,\text{Converted})}(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{\pi})}{\partial \boldsymbol{w}_{m,d}} = \frac{1}{\sqrt{M}} \boldsymbol{x}_i \dot{\sigma}\left(w_{m,d,k_d} x_{i,k_d}\right) \prod_{s \in \{1,2,3,\ldots,D\} \setminus \{d\}} \sigma\left(w_{m,s,k_s} x_{i,k_s}\right) \sum_{\ell=1}^{2^D} \pi_{m,\ell}. \quad \text{(AAI)} \tag{C.3}$$

Since $\pi_{m,\ell}$ is initialized as zero-mean i.i.d. Gaussians with unit variances,

$$\mathbb{E}_m\left[\pi_{m,\ell}\pi_{m,\ell'}\right] = \begin{cases} 0 & \text{if } \ell \neq \ell', \\ 1 & \text{otherwise.} \end{cases} \tag{C.4}$$

Therefore,

$$\mathbb{E}_m\left[\left(\pi_{m,1} + \pi_{m,2} + \cdots + \pi_{m,2^D}\right)^2\right] = 2^D. \tag{C.5}$$

The limiting NTK contribution from the parameters at depth d is

$$\lim_{M\to\infty} \frac{1}{M} \sum_{m=1}^{M} \Bigg(\Sigma_{\{i,j\},k_d} \times \dot{\sigma}(w_{m,d,k_d} x_{i,k_d}) \dot{\sigma}(w_{m,d,k_d} x_{j,k_d})$$

$$
\begin{aligned}
&\times \prod_{s\in\{1,2,3,\dots,D\}\setminus\{d\}} \sigma\big(w_{m,s,k_s}x_{i,k_s}\big) \prod_{s\in\{1,2,3,\dots,D\}\setminus\{d\}} \sigma\big(w_{m,s,k_s}x_{j,k_s}\big) \\
&\times \sum_{\ell=1}^{2^D}\pi_{m,\ell}\sum_{\ell=1}^{2^D}\pi_{m,\ell}\Big) \\
=&\Sigma_{\{i,j\},k_d} \times \mathbb{E}_m\left[\underbrace{\dot\sigma\big(w_{m,d,k_d}x_{i,k_d}\big)\dot\sigma\big(w_{m,d,k_d}x_{j,k_d}\big)}_{\to\dot{\mathcal{T}}_{\{i,j\},k_d}}\right] \\
&\times \mathbb{E}_m\left[\underbrace{\prod_{s\in\{1,2,3,\dots,D\}\setminus\{d\}} \sigma\big(w_{m,s,k_s}x_{i,k_s}\big) \prod_{s\in\{1,2,3,\dots,D\}\setminus\{d\}} \sigma\big(w_{m,s,k_s}x_{j,k_s}\big)}_{\to\prod_{t\in h(a)\setminus\{k_d\}}\mathcal{T}_{\{i,j\},t}}\right] \\
&\times \mathbb{E}_m\left[\underbrace{\sum_{\ell=1}^{2^D}\pi_{m,\ell}\sum_{\ell=1}^{2^D}\pi_{m,\ell}}_{\to 2^D}\right], \qquad \text{(C.6)}
\end{aligned}
$$

where a is the root-to-leaf path which is common for all leaves in an oblivious tree. Since there are D possible locations for d, we obtain

$$
\Theta^{(D,\text{Oblivious, nodes})}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \Theta^{(D,\text{Converted, nodes})}(\boldsymbol{x}_i,\boldsymbol{x}_j) = 2^D \sum_{s\in h(a)}\left(\Sigma_{\{i,j\},s}\dot{\mathcal{T}}_{\{i,j\},s}\prod_{t\in h(a)\setminus\{s\}}\mathcal{T}_{\{i,j\},t}\right). \qquad \text{(C.7)}
$$

For leaves at initialization,

$$
\frac{\partial f^{(D,\text{Converted})}(\boldsymbol{x}_i,\boldsymbol{w},\boldsymbol{\pi})}{\partial \pi_{m,\ell}} = \frac{1}{\sqrt{M}}\prod_{s\in\{1,2,3,\dots,D\}}\sigma\big(w_{m,s,k_s}x_{i,s}\big). \qquad \text{(C.8)}
$$

Since there are 2^D leaves, we obtain

$$
\Theta^{(D,\text{Oblivious, leaves})}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \Theta^{(D,\text{Converted, leaves})}(\boldsymbol{x}_i,\boldsymbol{x}_j) = 2^D\prod_{s\in h(a)}\mathcal{T}_{\{i,j\},s}. \qquad \text{(C.9)}
$$

Summation of Equation (C.7) and (C.9) is consistent to Equation (A.1) with perfect binary tree architecture that use the same feature at each depth.

Up until this point, our discussion has focused on the scenario where $\beta = 0$. Since we can incorporate the bias term by introducing an additional feature that consistently takes the value β as its input, the contribution of the bias term can be represented by adding β^2 to the inner product of the inputs in $\Sigma_{\{i,j\},s}$ when $\beta \neq 0$. □

D. Computational Complexity of MKL procedures

First, we analyze the computational cost of kernel matrix computation. To obtain a single kernel matrix, a calculation defined in Equation (7) is performed N^2 times, where N is the size of an input dataset. When we denote the number of leaves as $\mathcal{L}$ and the depth of the tree as D, the overall computational complexity is $\mathcal{O}(N^2\mathcal{L}D^2)$. Note that if there are duplicates in the output of $h(a_\ell)$ for all $\ell \in [\mathcal{L}]$, the computational cost can be reduced by aggregating and computing these duplicates together, so the actual computational cost is often less than this. For example, when considering oblivious trees, the computational complexity reduces to $\mathcal{O}(N^2D^2)$. Second, we consider the computational cost of MKL. Since MKL uses multiple kernels, it repeats the calculation of kernel matrices for all the kernel matrices, while parallelization is possible in this process. The cost to calculate the weights of the kernels by EasyMKL depends on the number of kernels. Specifically, the EasyMKL uses Kernelized Optimization of the Margin Distribution (KOMD) (Aiolli et al., 2008), and its computational cost is known to be linear with respect to the number of kernels (Aiolli & Donini, 2015).

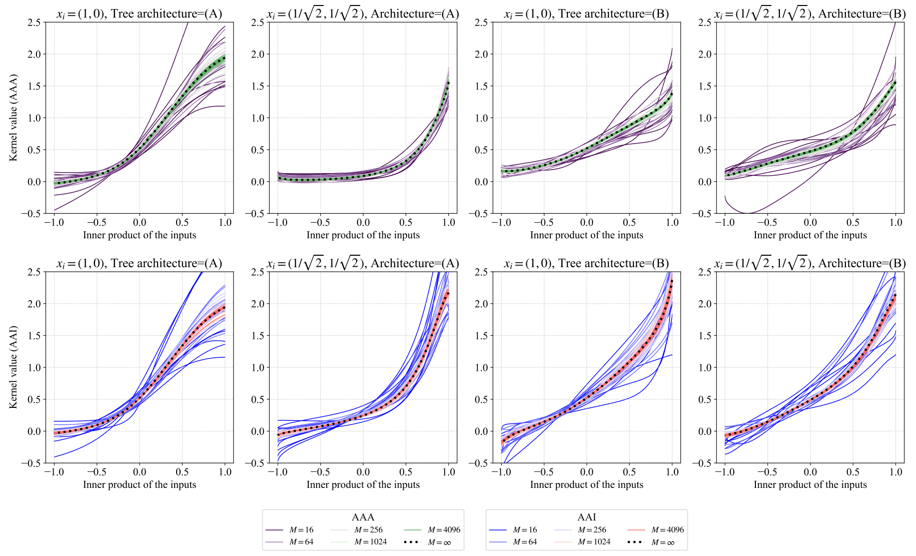

Figure A.2. An empirical demonstration of convergence of $\widehat{\Theta}_0(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to the fixed limit $\Theta(\boldsymbol{x}_i, \boldsymbol{x}_j)$ as M increases. Two conditions (AAA/AAI) are listed vertically, and settings of vectors and tree architectures for computing the kernel are listed horizontally.

E. Additional Experiments

The implementation we used in our numerical experiments is available online[1].

E.1. Convergence of the NTK with AAA/AAI

Figure A.2 shows the convergence of the NTK with AAA or AAI cases as the number M of trees increases on the same datasets and tree architectures used in Figure 2. We set $\alpha = 2.0$ and $\beta = 0.5$. The kernels induced by finite trees $M = \{16, 64, 256, 1024, 4096\}$ are computed numerically by re-initializing the parameters 10 times. We plot two cases: $\boldsymbol{x}_i = (1, 0), \boldsymbol{x}_j = (\cos(\omega), \sin(\omega))$ with $\omega \in [0, \pi]$, and $\boldsymbol{x}_i = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}), \boldsymbol{x}_j = (\cos(\omega), \sin(\omega))$ with $\omega \in [\frac{\pi}{4}, \frac{5\pi}{4}]$. We employ a perfect binary tree of depth 2. For architecture (A), the first feature is used at both depths 1 and 2. For architecture (B), the first feature is used at depth 1 and the second feature at depth 2. This visualization confirms that as the number of trees increases, the kernel asymptotically approaches the formula defined in Theorem 3.1.

E.2. Visualization of the NTK with AAA/AAI for each hyperparameter

We performed the same visualization as in Figure 2 with varying hyperparameters. The results are shown in Figures A.3, A.4, A.5, and A.6.

E.3. Output dynamics with a real-world dataset

Figure A.7 demonstrates that for both AAA and AAI, as the number of trees increases, the trajectory derived analytically from the limiting kernel becomes more aligned with the trajectory observed during gradient descent training. The protocol used to create this figure is the same as that for Figure 3. In this experiment, we used the diabetes dataset[2], a commonly used real-world dataset for regression tasks that predicts a quantitative measure of disease progression one year after the baseline. The diabetes dataset consists of $F = 10$ features, including body mass index, average blood pressure, age, sex, and six blood serum measurements. All features and prediction targets have been standardized to have zero mean and unit variance. We considered an ensemble of perfect binary trees with parameters $\alpha = 2.0$ and $\beta = 0.5$. The body mass index was chosen for splitting at depth 1, while the average blood pressure was used for depth 2 during initialization. We selected 50 random training samples and 10 test samples for this study.

[1]https://github.com/ryuichi0704/aa-tntk
[2]https://archive.ics.uci.edu/dataset/34/diabetes

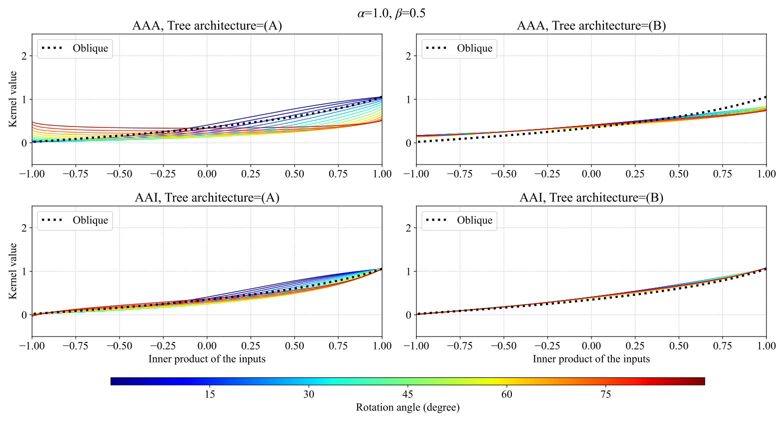

Figure A.3. The rotation angle dependency of $\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $\alpha = 1.0$ and $\beta = 0.5$. The protocol for creating the figure is the same as Figure 2.

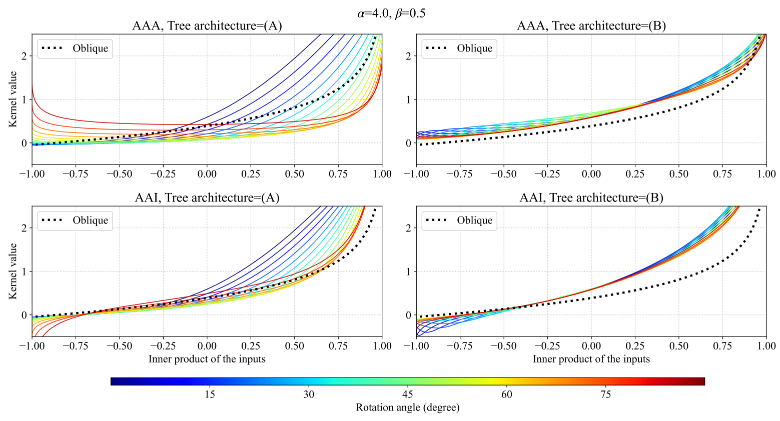

Figure A.4. The rotation angle dependency of $\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $\alpha = 4.0$ and $\beta = 0.5$. The protocol for creating the figure is the same as Figure 2.

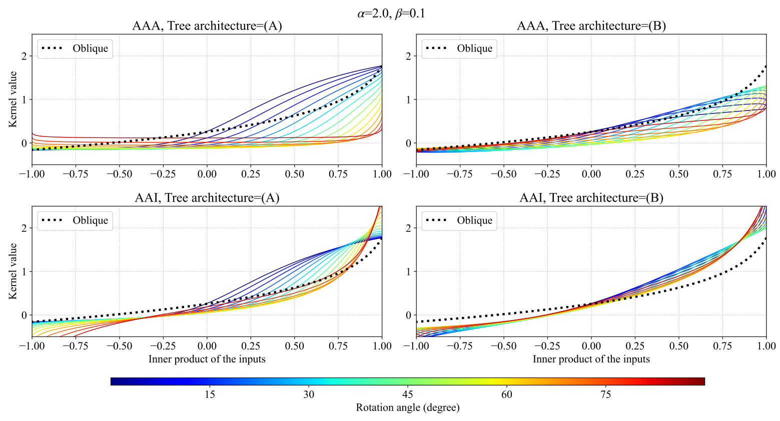

Figure A.5. The rotation angle dependency of $\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $\alpha = 2.0$ and $\beta = 0.1$. The protocol for creating the figure is the same as Figure 2.

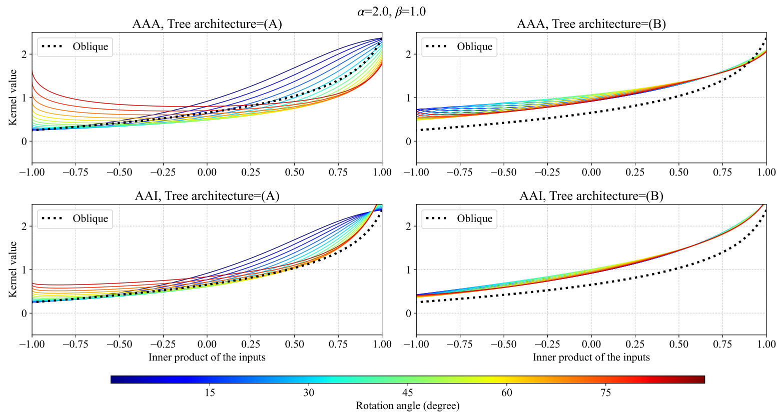

Figure A.6. The rotation angle dependency of $\Theta^{\text{AxisAligned}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $\alpha = 2.0$ and $\beta = 1.0$. The protocol for creating the figure is the same as Figure 2.

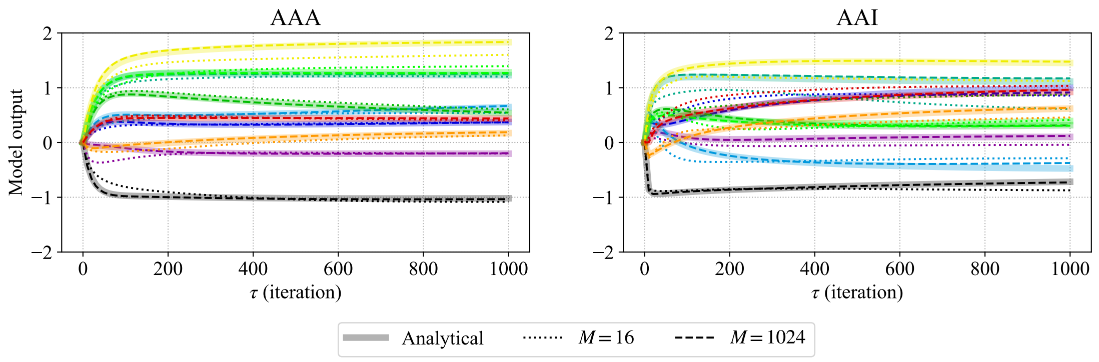

Figure A.7. Output dynamics of test data points for axis-aligned soft tree ensembles with two conditions. The protocol used to create the figure is identical to that of Figure 3.

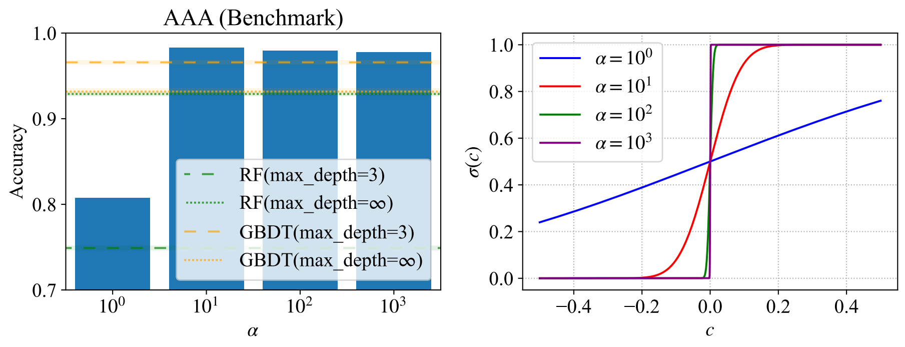

Figure A.8. Accuracy of AAA (Benchmark) on the tic-tac-toe dataset when varying α, with $\beta = 0.5$.

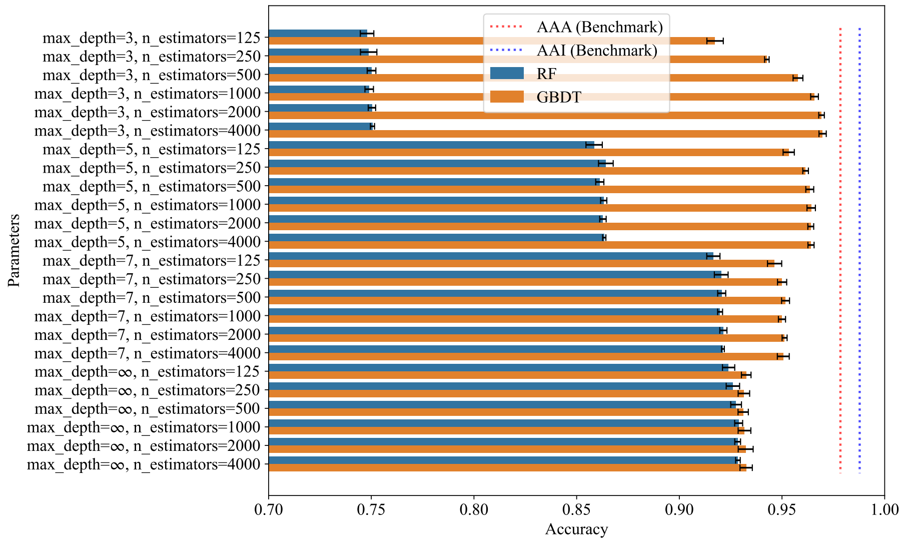

Figure A.9. Performance of RF and GBDT on the tic-tac-toe dataset for each `max_depth` and `n_estimators`. The procedure of the experiment is the same as that in Figure 7.

E.4. Comparison with RF and GBDT

We further examine the performance of RF and GBDT on the tic-tac-toe dataset and discuss the effectiveness of AAA reported in Figure 7.

Even when feature selection is not explicitly conducted ("AAA (Benchmark)" in Figure 7), the performance of the AAA model calculated using the NTK is superior to that of a typical RF and GBDT. In addition, as shown in Figure A.8, even if α increases to the extent that the splitting function is nearly equivalent to a step function, the performance of AAA (Benchmark) remains superior to that of RF and GBDT. These results suggest that factors other than the feature selections and the softness of the splitting are important. This comparison of AAA (Benchmark) and RF/GBDT means that such a gradient descent-based learning is more effective than a greedy learning approach of RF/GBDT for the tic-tac-toe dataset. The output variable in the tic-tac-toe dataset, the game's outcome, is determined only by the third-order interactions of features, and it seems that greedy approaches are not appropriate to pick up such interactions.

Moreover, as shown in Figure A.9, even when `max_depth` and `n_estimators` are tuned, the performance of RF and GBDT does not reach that of AAA (Benchmark). In terms of trained models, if the splitting function is replaced with a step function, AAA, RF, and GBDT all have the same format, which indicates that there is no difference in the expressive power due to tree architectures. These observations indirectly support the fact that how to learn parameters in the tree is a contributing factor.

E.5. MKL weights obtained from various datasets

To investigate how MKL behaves on datasets other than the tic-tac-toe dataset, we selected and used additional 14 binary classification datasets from Fernández-Delgado et al. (2014) with fewer than 1000 data points and 10 features. See Table A.1 for details. The literature on kernel methods often uses smaller datasets are often used due to the high computational cost. However, it is known that such small domains have become important for leveraging NTKs (Arora et al., 2020). We constructed kernels considering interactions up to the second order, resulting in $\binom{F}{1} + \binom{F}{2}$ kernels. Figure A.10 shows the weights obtained in a manner similar to Figure 6. Similar to the tic-tac-toe dataset, AAI yields weights that are relatively close to a uniform distribution, while AAA tends to produce larger weights for specific interactions. To quantitatively analyze such trends, we compared the KL (Kullback-Leibler) divergence between the obtained weights and a uniform distribution to examine how it behaves under AAA or AAI. Results are shown in Figure A.11. From this figure, it can be observed that the KL divergence for AAA is larger, indicating a tendency to deviate from a uniform distribution, and this holds true across various datasets.

Table A.1. Summary of dataset characteristics for binary classification.

Name	N (Instances)	F (Features)
acute-inflammation	120	6
acute-nephritis	120	6
balloons	16	4
blood	748	4
breast-cancer	286	9
breast-cancer-wisc	699	9
echocardiogram	131	10
fertility	100	9
haberman-survival	306	3
ilpd-indian-liver	583	9
mammographic	961	5
pima	768	8
pittsburg-bridges-T-OR-D	102	7
tic-tac-toe	958	9
vertebral-column-2clases	310	6

E.6. Generalization performance on various datasets

All the results about the generalization performances are shown in Figures A.12, A.13, A.14, and A.15. As shown in Figure A.12, the accuracy trend on the tic-tac-toe dataset as shown in Figure 7 in the main text does not largely depend on β. This trend is observed in a variety of datasets, which are also used in Appendix E.5, as shown in Figures A.13, A.14, and A.15. Overall, the performance depends on datasets and it is not fundamental to make a general claim that AAA and AAI are better or worse than other models in terms of generalization performance compared to other models. This is a natural consequence and orthogonal to our claim in this paper.

F. Limitations

NTK-based analyses have limitations, being effective in specific conditions like lazy training (Chizat et al., 2019). Therefore, a theoretical gap still exists between NTK-based analyses and practical models.

In addition, the positive definiteness of the kernel depends on the combination of the dataset and the features used for splitting. Under certain conditions, such as when only a single feature is used for splitting and this feature is insufficient to fully separate the data, the kernel may not maintain its positive definiteness and may become positive semidefinite. Although it is possible to address this by deepening the decision tree or using a variety of tree architectures to enhance the features used for splitting, we need to note that positive definiteness is not always guaranteed. Upon numerically verifying whether the kernel is positive definite across the 15 datasets used in Table A.1, excluding duplicate records and considering up to third-order interactions, both AAA (Benchmark) and AAI (Benchmark) are found to be positive definite for all folds of the all dataset. When interactions are limited up to the second order, rank deficiency occurred in the tic-tac-toe, breast-cancer-wisc, and mammographic datasets.

Applying a temperature-scaled entmax for weights $\boldsymbol{w}_{m,n}$ might be an interesting formulation for selecting features used for splitting during training, although it is not addressed in this paper due to the difficulty of deriving a closed NTK (Popov et al., 2020; Chang et al., 2022). We will leave these challenges for future investigation.

G. Relationship to the Multi-Layer Perceptron

NTK-based analysis using the $1/\sqrt{M}$ scaling defined in Equation (3), primarily targeting neural networks, has produced various insights. We argue that the tree structure of a model does not mean its situation is special compared to neural

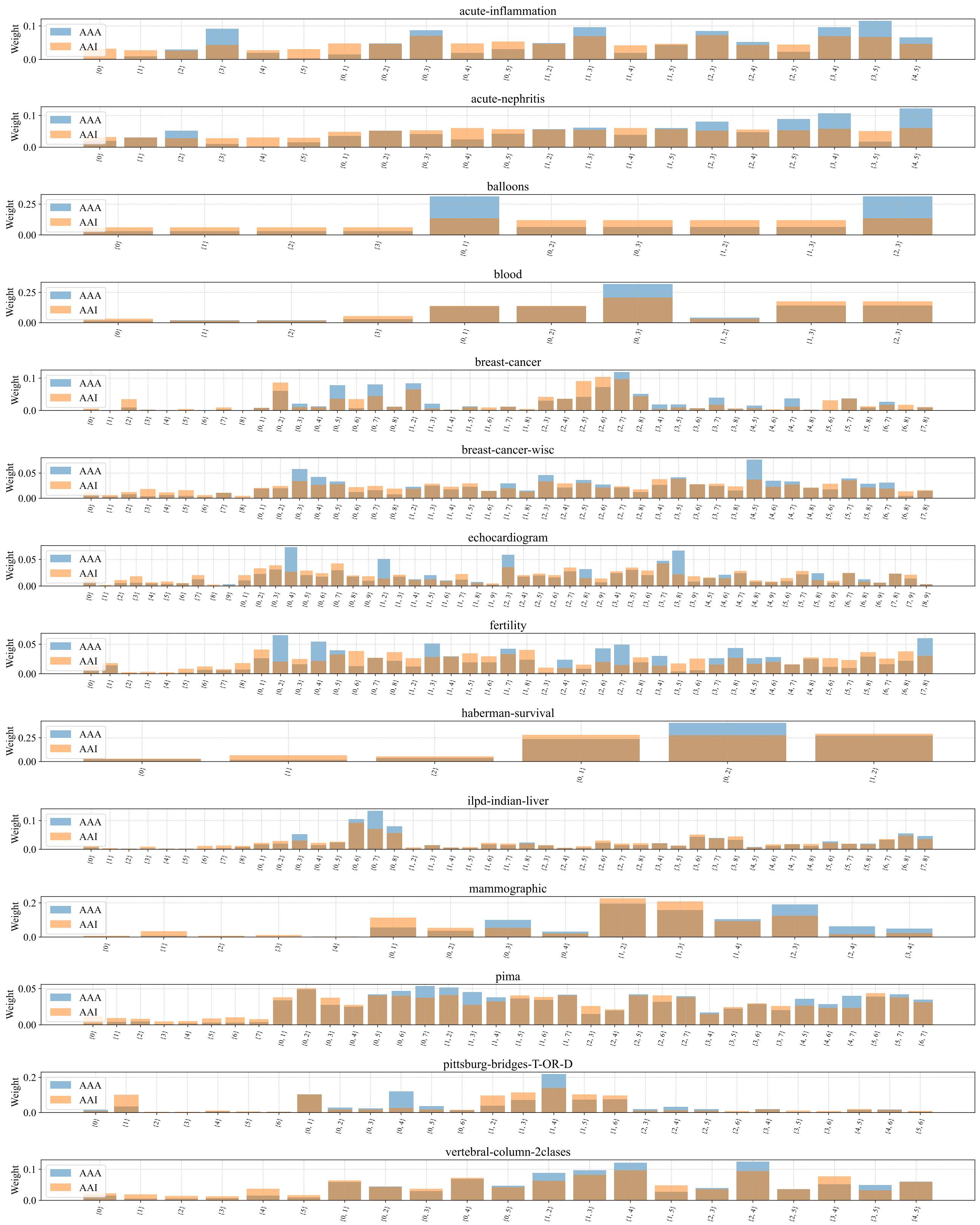

Figure A.10. Weights of a linear combination of multiple kernels obtained by EasyMKL on 14 UCI dataset.

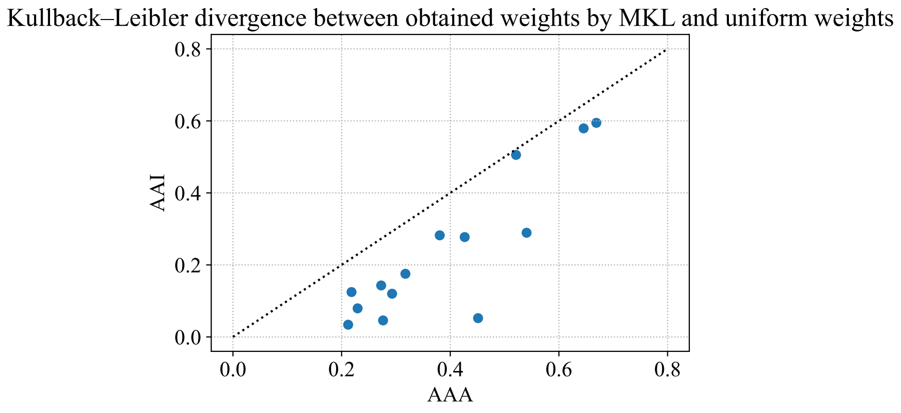

Figure A.11. The KL divergence from the weights obtained by MKL to the uniform distribution under AAA or AAI. Each point on the scatter plot corresponds to a specific dataset.

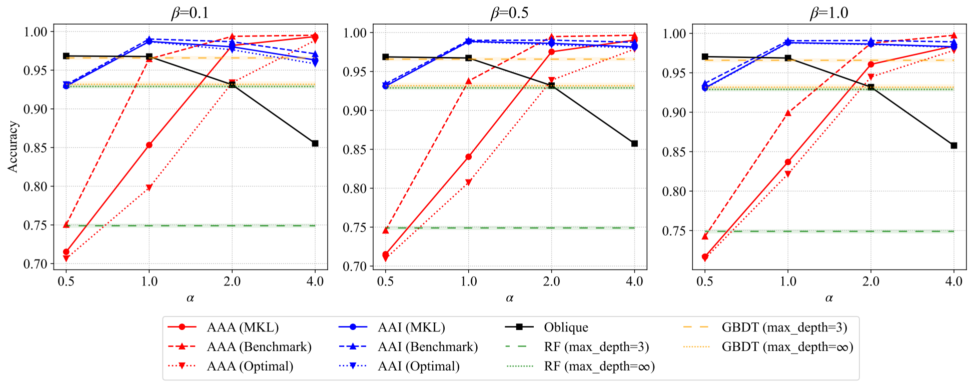

Figure A.12. Classification accuracy on the tic-tac-toe dataset with $\beta = \{0.1, 0.5, 1.0\}$. The procedure of the experiment is the same as that in Figure 7.

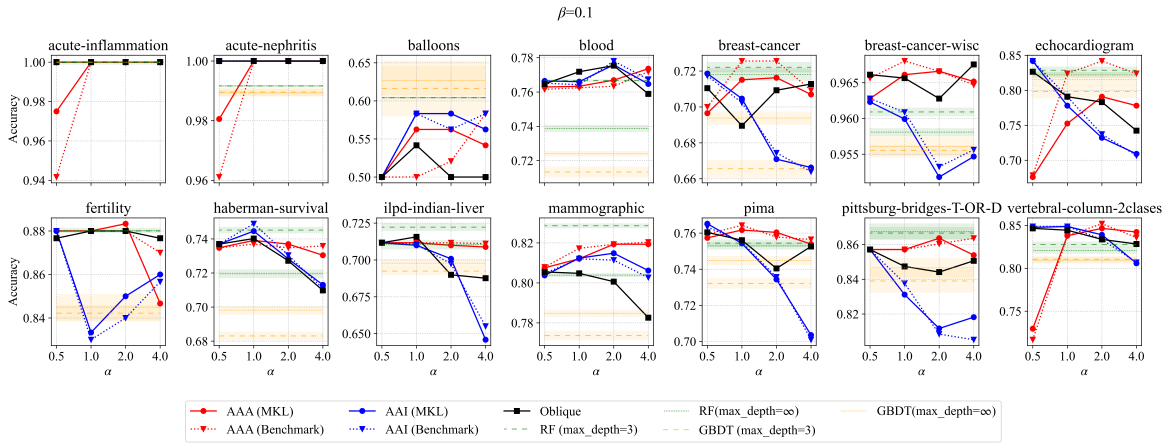

Figure A.13. Classification accuracy on 14 UCI dataset with $\beta = 0.1$. The procedure of the experiment is the same as that in Figure 7. Interactions are considered up to the second order.

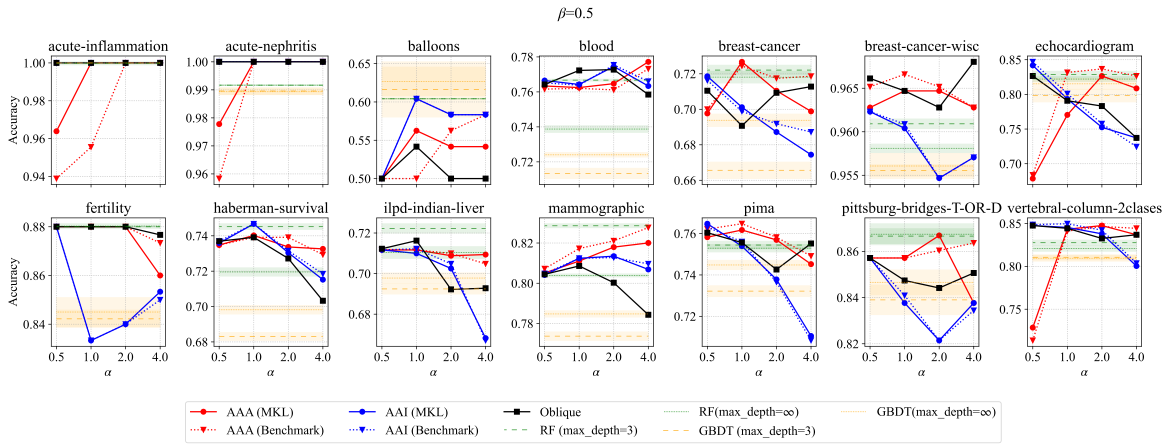

Figure A.14. Classification accuracy on 14 UCI dataset with $\beta = 0.5$. The procedure of the experiment is the same as that in Figure 7. Interactions are considered up to the second order.

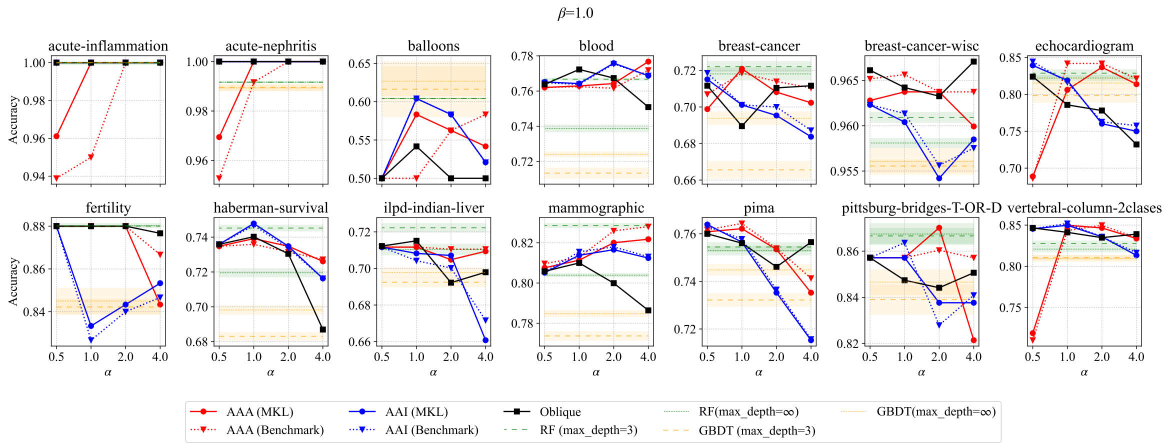

Figure A.15. Classification accuracy on 14 UCI dataset with $\beta = 1.0$. The procedure of the experiment is the same as that in Figure 7. Interactions are considered up to the second order.

networks, as there are clear correspondences between Multi-Layer Perceptron (MLP) and tree ensemble models. The formulation of a tree ensemble with depth 1 is:

$$\begin{aligned} f(\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{\pi}) &= \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left(\sigma\left(\boldsymbol{w}_{m,1}^{\top} \boldsymbol{x}_i + \beta b_{m,1}\right) \pi_{m,1} + \left(1 - \sigma\left(\boldsymbol{w}_{m,1}^{\top} \boldsymbol{x}_i + \beta b_{m,1}\right)\right) \pi_{m,2}\right) \\ &= \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left(\left(\pi_{m,1} - \pi_{m,2}\right) \sigma\left(\boldsymbol{w}_{m,1}^{\top} x_i + \beta b_{m,1}\right) + \pi_{m,2}\right). \end{aligned} \tag{G.1}$$

When we consider the correspondence between $\pi_{m,1} - \pi_{m,2}$ in tree ensembles and second layer weights in the two-layer perceptron, the tree ensembles model coincides with the two-layer perceptron. Making the tree deeper corresponds to making the first layer of the two-layer perceptron more complex. Therefore, NTK-based analysis for trees is an extension of the NTK-based analysis that has been conducted on neural networks.