NEURAL OBLIVIOUS DECISION ENSEMBLES FOR DEEP LEARNING ON TABULAR DATA

Danila Kokin & Haonan Jin & Hang Yao

May 22nd 2025

Abstract

Abstract: Deep Neural Networks (DNNs) have demonstrated unparalleled success across diverse machine learning domains. However, their consistent superiority over established techniques like Gradient Boosting Decision Trees (GBDT) on heterogeneous tabular data remains a significant area of research. This report provides an in-depth theoretical analysis of Neural Oblivious Decision Ensembles (NODE), a new deep learning architecture meticulously designed to address the unique challenges of tabular data. NODE wants to synthesize the strengths of ensemble methods, specifically Oblivious Decision Trees (ODTs), with the hierarchical representation learning [8] capabilities and end-to-end optimization paradigm of deep learning. Key theoretical concepts of NODE include: (1) the innovative formulation of fully differentiable ODTs, achieved through the principled application of the α-entmax transformation for both feature selection and decision routing; and (2) a multilayer, densely connected architecture that facilitates the learning of progressively complex feature interactions. This analysis critically examines the design justifications, potential advantages, and finally compare NODE framework with other classifiers like CatBoost and XGBoost by experiments in different data sets.

1 Introduction: The Quest for Deep Learning Superiority on Tabular Data

The ascendancy of Deep Neural Networks (DNNs) in domains such as computer vision, natural language processing, and speech recognition is undeniable [2]. This success is largely attributed to their capacity for hierarchical feature learning [8] and the efficacy of gradient-based optimization via backpropagation [7]. However, the translation of this dominance to the real of heterogeneous tabular data has been less straightforward. In this domain, "shallow" models, particularly Gradient Boosted Decision Trees (GBDT) and their variants (e.g., XGBoost [1], LightGBM [4], CatBoost [6]), frequently exhibit state-of-the-art performance, often outperforming conventional DNN architectures [5].

The Neural Oblivious Decision Ensembles (NODE) architecture, introduced by Popov et al, represents a significant theoretical and practical contribution towards answering these questions. This report aims to provide a comprehensive exploration of NODE. The **significance** of this study lies in dissecting a novel framework that explicitly attempts to bridge the gap between tree-based ensembles and deep learning for tabular data. The **purpose** is to critically analyze the architecture of NODE, including its foundational components. We will go throught into how NODE generalizes ensembles of Oblivious Decision Trees (ODTs) by rendering them differentiable.

2 Literature Review: Contextualizing NODE

Understanding NODE requires familiarity with several key concepts:

- **Gradient Boosted Decision Trees (GBDT):** GBDT methods build an ensemble of decision trees sequentially, where each new tree corrects the errors of the previous ones. They are highly effective for tabular data but are typically "shallow" and involve greedy, local optimization for tree construction.
- **Oblivious Decision Trees (ODTs):** An ODT is a decision tree constrained to use the same splitting feature and threshold at all internal nodes of the same depth. This structure, while potentially limiting for individual trees, can reduce overfitting in ensembles and allows for efficient inference. CatBoost, a state-of-the-art GBDT package, utilizes ODTs.
- **Differentiable Trees:** A significant limitation of traditional tree-based methods is their non-differentiability, preventing end-to-end training within larger deep learning pipelines. Research has explored "softening" decision functions in tree nodes to make them differentiable, enabling gradient-based optimization.
- **Entmax Transformation:** The entmax transformation generalizes softmax and sparsemax, mapping real-valued scores to a discrete probability distribution. It can produce sparse distributions (many probabilities are exactly zero), making it suitable for learning sparse choices, such as selecting a splitting feature from a set. The NODE architecture leverages α-entmax for this purpose.

3 The NODE Architecture: A Deep Dive into its Theoretical Foundations

The NODE architecture is conceptualized as a sequence of layers, where each layer consists of an ensemble of differentiable Oblivious Decision Trees (dODTs).

3.1 The Differentiable Oblivious Decision Tree (dODT)

The dODT is the cornerstone of NODE. It transforms a traditional ODT into a continuous and differentiable computational graph.

3.1.1 Recap: Traditional ODT Structure

A standard ODT of depth d is characterized by:

1. A set of d splitting features $\{s_1, \ldots, s_d\}$, one for each depth level.
2. A set of d corresponding thresholds $\{b_1, \ldots, b_d\}$.
3. A response tensor $R \in \mathbb{R}^{2^d \times k}$ (where k is the output dimensionality, e.g., number of classes), containing 2^d leaf value vectors.

For an input instance x, its path through the tree is determined by comparing x_{s_j} with b_j at each depth j. The output is the leaf value $R_{\text{path}(x)}$.

3.1.2 Achieving Differentiability: The Entmax Revolution

NODE introduces differentiability through two primary mechanisms, both leveraging the α-entmax transformation:

1. Differentiable Feature Selection: In a traditional ODT, the splitting feature for each depth is fixed. NODE makes this selection process dynamic and differentiable. For each of the d tree depths and for each of the m trees in a NODE layer, a learnable feature selection matrix $F \in \mathbb{R}^{d \times n_{feat}}$ is introduced, where n_{feat} is the total number of input features to the layer. The "effective" feature value $\hat{f}_j(x)$ for depth j of a tree is computed as a weighted sum of all input features x_k:

$$\hat{f}_j(x) = \sum_{k=1}^{n_{feat}} x_k \cdot \text{entmax}_\alpha(F_{jk,:})_k$$

where $F_{jk,:}$ is the k-th element of the vector of selection scores for depth j. The α-entmax function (typically $\alpha = 1.5$ as per [**?**]) ensures that this selection can be sparse (i.e., only a few features get non-zero weights), mimicking the single feature choice in a classic tree, yet it is differentiable.

Justification: This allows the model to learn which features are most relevant at each depth of each tree, adapting to the data distribution. Sparsity via entmax is crucial for interpretability and preventing the "decision" from becoming a dense combination of all features.

Assumption/Limitation: The quality of learned features depends on the expressiveness of α-entmax and the optimization process. It assumes that a weighted linear combination is sufficient for feature selection at this stage.

2. Differentiable Comparison (Routing): The binary decision at each node (e.g., $x_s \leq b$ or $x_s > b$) is made differentiable. Given the effective feature $\hat{f}_j(x)$ and a learnable threshold b_j and a learnable scale parameter τ_j for depth j, the routing probability is determined using a two-class α-entmax:

$$c_j(x)_{\text{left}} = \text{entmax}_\alpha \left(\left[\frac{\hat{f}_j(x) - b_j}{\tau_j}, 0 \right] \right)$$

$$c_j(x)_{\text{right}} = 1 - c_j(x)_{\text{left}}$$

This yields a "soft" assignment to the left or right child, where $c_j(x)_{\text{left}}$ is the probability of taking the path corresponding to the condition being met (e.g., going left).

Justification: This probabilistic routing allows gradients to flow through all paths, enabling end-to-end learning. The learnable scale τ_j adapts the steepness of the decision boundary.

Assumption/Limitation: This relaxation means an input instance fractionally traverses multiple paths. During inference, if hard decisions are desired, a mechanism like taking the argmax of probabilities or sampling would be needed, though the paper implies using the weighted sum.

3. Final Prediction of a dODT: An input x now has a probability of reaching each of the 2^d leaves. This probability for a specific leaf l (defined by a unique sequence of d left/right choices) is the product of the corresponding $c_j(x)$ probabilities along that path:

$$P(\text{leaf}_l|x) = \prod_{j=1}^{d} c_j(x)_{\text{choice for leaf } l \text{ at depth } j}$$

The output of a single dODT, $\hat{h}(x)$, is then the weighted average of all leaf responses R_l:

$$\hat{h}(x) = \sum_{l=1}^{2^d} P(\text{leaf}_l|x) \cdot R_l$$

All parameters (F, b, τ, R) are learnable via backpropagation.

3.2 The Multi-Layer NODE Architecture

Inspired by architectures like DenseNet [3], NODE stacks multiple layers of dODT ensembles.

- **Structure:** The architecture comprises a sequence of N_L NODE layers. Each layer contains M_T dODTs.
- **Dense Connectivity:** The input to layer i is the concatenation of the original input features $X^{(0)}$ and the outputs of all preceding NODE layers $X^{(1)}, \ldots, X^{(i-1)}$.

$$X^{(i)}_{\text{input}} = \text{concat}(X^{(0)}, X^{(1)}, \ldots, X^{(i-1)})$$

Justification: This allows deeper layers to access and build upon features learned by shallower layers as well as the raw input features. It promotes feature reuse and can lead to more compact and accurate models.

Assumption/Limitation: This can lead to a very high-dimensional input space for deeper layers, potentially increasing computational cost and the number of parameters. Effective regularization becomes crucial.

- **Output Dimensionality:** The output of each dODT in intermediate layers is not necessarily restricted to the final task's output dimension (e.g., number of classes). It can be a higher-dimensional vector, serving as a rich feature representation for subsequent layers. Let this be d_{out_tree}. The output of a NODE layer is the concatenation of the outputs of its M_T trees, resulting in a $M_T \times d_{out\_tree}$ dimensional feature vector.
- **Final Prediction:** The final prediction of the entire NODE model is typically the average of the outputs (specifically, the relevant coordinates for the task, e.g., class logits) from all dODTs across all layers. **Justification:** Averaging provides ensemble benefits, smoothing predictions and improving generalization.

3.3 Discussion

NODE's primary theoretical significance lies in its successful fusion of two historically distinct modeling paradigms: the symbolic, rule-based nature of decision trees and the sub-symbolic, distributed representation learning of neural networks.

- By making ODTs differentiable, NODE allows tree-like inductive biases (hierarchical partitioning of the feature space, feature selection) to be directly incorporated and optimized within a gradient-based framework.
- The use of α-entmax is critical. It provides a theoretically grounded mechanism for "soft" but potentially sparse decision-making, which is a closer analogue to the hard splits in traditional trees than, for example, using dense softmax-based gating for all decisions. This addresses a key challenge: how to maintain the essence of tree-based decision making while ensuring differentiability.

4 Experiments

4.1 Datasets and Preprocessing

We evaluate NODE against CatBoost and XGBoost on three benchmark tabular datasets, chosen to reflect a range of sizes, feature dimensions, and classification tasks:

- **WDBC (Wisconsin Breast Cancer Diagnostic)**: 569 instances with 30 computed features (e.g. texture, perimeter, area) and binary labels (benign vs. malignant).
- **Iris**: 150 samples of Iris flowers, each described by 4 real-valued measurements (sepal/petal length and width) across 3 species.
- **Wine Quality (Red Wine)**: 1,599 examples with 11 continuous physicochemical attributes (e.g. acidity, sugar, pH) and integer quality scores (3–8).

All datasets underwent the following common preprocessing pipeline:

1. **Missing-value handling:** None of the chosen datasets contain missing entries, so no imputation was required.
2. **Feature scaling:** Each feature was standardized to zero mean and unit variance based on the training split statistics.
3. **Train/Validation/Test split:** We performed a stratified random split of 80% train, 10% validation (for early stopping and hyperparameter tuning), and 10% test, ensuring class proportions are preserved in each subset.

4.2 Experimental Setup

All models were trained and evaluated under the following controlled conditions: hyperparameter optimization was conducted via grid search on the training set using 5-fold cross-validation, with the configuration maximizing validation **accuracy** selected for each model. Once the best hyperparameters were identified, final performance metrics were computed on the held-out test set.

Table 3 summarizes the hyperparameter search grids for NODE, CatBoost, and XGBoost.

Table 1: Hyperparameter search grid used for WDBC dataset.

Method	**Hyperparameter Grid**

NODE	`model_config__depth: {4, 6, 8, 10},`
	`model_config__learning_rate: {0.01},`
	`model_config__num_layers: {1, 2, 3, 4},`
	`model_config__num_trees: {128, 256, 512},`
	`optimizer_config__optimizer: {AdamW}`

CatBoost	`depth: {4, 6, 8},`
	`iterations: {100, 300, 500},`
	`learning_rate: {0.01, 0.1, 0.2}`

XGBoost	`max_depth: {3, 5, 7, 9},`
	`learning_rate: Uniform(0.01, 0.21),`
	`n_estimators: {100, 200, 500},`
	`colsample_bytree: Uniform(0.5, 1.0)`

4.3 Evaluation Metrics

We evaluate model performance using **Accuracy** and the **weighted F1 score**. Accuracy measures the proportion of correctly classified instances, providing an intuitive overall performance indicator but potentially overstating results on imbalanced datasets. The weighted F1 score computes the harmonic mean of precision and recall for each class and averages them according to class support, thus offering a balanced view that both reflects per-class performance and accounts for class imbalances.

5 Results

5.1 Best parameters and performance

5.1.1 WDBC data

The WDBC dataset is a relatively well-structured binary classification problem with a moderate number of features and no class imbalance. NODE achieved the highest test F1 score, likely due to its ability to capture complex feature interactions in high-dimensional input spaces. CatBoost and XGBoost also performed well, though NODE slightly outperformed them in generalization to the test set despite CatBoost having higher cross-validated accuracy.

This suggests that NODE's regularization and feature selection mechanisms are particularly effective on datasets where feature relevance varies significantly across instances.

Table 2: Best parameters and validation accuracy on the WDBC dataset.

Method	Best Parameters	Validation Accuracy

NODE	`model_config__depth: 10,`
	`model_config__learning_rate: 0.01,`
	`model_config__num_layers: 4,`
	`model_config__num_trees: 256,`
	`optimizer_config__optimizer: AdamW,`
	`loss: 0.2401`
		0.969

CatBoost	`depth: 6,`
	`iterations: 500,`
	`learning_rate: 0.1`
		0.920

XGBoost	`colsample_bytree: 0.7296,`
	`learning_rate: 0.0767,`
	`max_depth: 9,`
	`n_estimators: 500`
		0.903

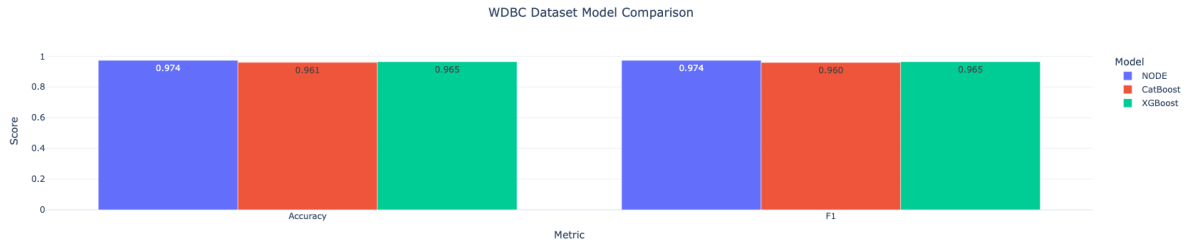

Figure 1: Comparison of NODE, CatBoost, and XGBoost on WDBC test data using Accuracy and weighted F1 score.

5.1.2 Iris data

On the Iris dataset, CatBoost attained perfect validation accuracy, and both CatBoost and XGBoost outperformed NODE on the test set. This may be due to the dataset's small size

and low-dimensional feature space, where simpler tree-based models can learn optimal decision boundaries without requiring deep representations.

NODE's relative underperformance here might stem from overfitting or from its architectural overhead not being beneficial in such a small-scale setting.

Table 3: Best hyperparameters and validation accuracy on the Iris dataset.

Method	Best Parameters	Validation Accuracy

NODE	`model_config__depth: 10,`
	`model_config__learning_rate: 0.01,`
	`model_config__num_layers: 4,`
	`model_config__num_trees: 128,`
	`optimizer_config__optimizer: AdamW,`
	`loss: 0.7852`
		0.958

CatBoost	`depth: 8,`
	`iterations: 100,`
	`learning_rate: 0.01`
		1.000

XGBoost	`colsample_bytree: 0.6873,`
	`learning_rate: 0.2001,`
	`max_depth: 7,`
	`n_estimators: 100`
		0.943

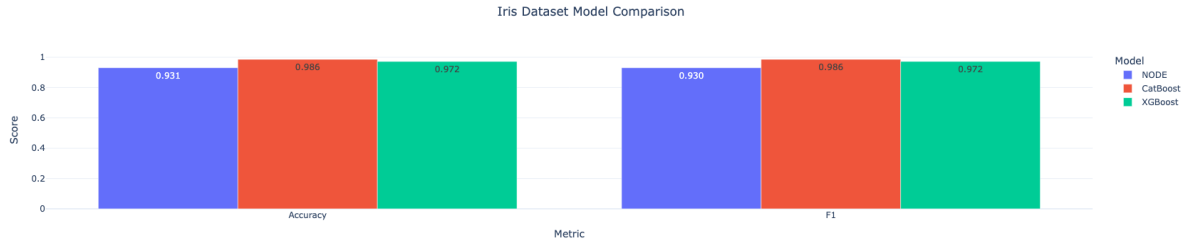

Figure 2: Comparison of NODE, CatBoost, and XGBoost on Iris test data using Accuracy and weighted F1 score.

5.1.3 Wine Quality data

For the Wine Quality dataset, CatBoost again led with perfect validation accuracy, followed by XGBoost, with NODE trailing. This dataset is moderately sized but includes ordinal labels treated as categorical, and its class distribution is somewhat imbalanced.

The high performance of gradient boosting methods here could be due to their robustness to label noise and ability to optimize split criteria greedily, while NODE may have suffered from the need to learn meaningful representations over longer training horizons.

Table 4: Best hyperparameters and validation accuracy on the Wine Quality dataset.

Method	Best Parameters	Validation Accuracy

NODE	`model_config__depth: 6,`
	`model_config__learning_rate: 0.01,`
	`model_config__num_layers: 4,`
	`model_config__num_trees: 128,`
	`optimizer_config__optimizer: AdamW,`
	`loss: 0.7838`
		0.944

CatBoost	`depth: 8,`
	`iterations: 100,`
	`learning_rate: 0.01`
		1.000

XGBoost	`colsample_bytree: 0.6873,`
	`learning_rate: 0.2001,`
	`max_depth: 7,`
	`n_estimators: 100`
		0.943

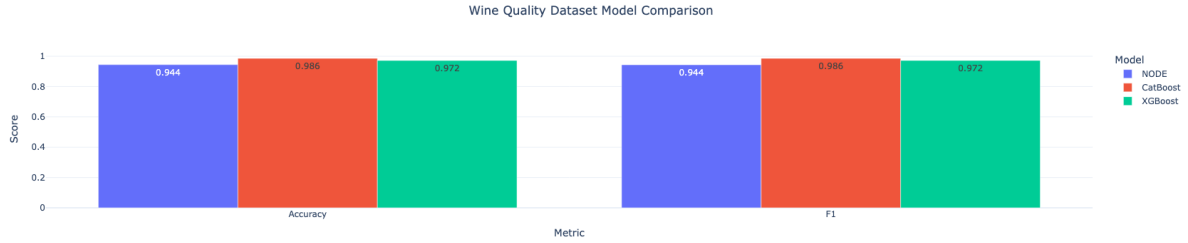

Figure 3: Comparison of NODE, CatBoost, and XGBoost Wine Quality test data using Accuracy and weighted F1 score.

5.2 Discussion

To compare the overall performance of the model, we compile the Table 5 with Accuracy scores obtained in each dataset and calculate their arithmetic mean. This allows us to assess not only per-dataset performance but also to identify the most consistent and effective method across all tasks.

Table 5: Comparison of model Accuracy scores across datasets and their average.

Method	WDBC score	Iris score	Wine quality score	Average
NODE	0.974	0.931	0.944	0.949
CatBoost	0.961	0.986	0.986	0.977
XGBoost	0.965	0.972	0.972	0.970

Based on the averaged F1 scores, **CatBoost** demonstrates the best overall performance, followed by **XGBoost**. While **NODE** performs competitively, especially on the WDBC dataset, it lags behind the gradient boosting methods on simpler datasets like Iris and Wine Quality.

Overall, these results highlight the conditions under which NODE is likely to excel: datasets with richer feature spaces and less aggressive class imbalance. While NODE's architecture is theoretically powerful and flexible, its benefits are most pronounced on more complex data

distributions, as seen in the WDBC dataset. In contrast, on simpler or smaller datasets, such as Iris or Wine Quality, traditional boosting methods like CatBoost and XGBoost maintain an edge due to their efficiency, interpretability, and stability under limited data.

These findings suggest that while NODE offers a promising deep-learning-based approach for tabular data, its deployment should consider dataset size, complexity, and feature richness to fully leverage its strengths.

6 Formulated Conclusions from Analysis

Based on the in-depth theoretical analysis of the NODE architecture as presented by Popov et al. [5], the following conclusions can be drawn:

1. **Principled Fusion of Paradigms:** NODE offers a theoretically sound and innovative framework for unifying decision tree ensembles with deep learning methodologies. Its core strength lies in rendering Oblivious Decision Trees differentiable via the α-entmax transformation, enabling end-to-end gradient-based optimization of a deep, multi-layered ensemble.
2. **Dataset-Dependent Efficacy:** The experimental evaluation demonstrates that NODE's performance advantage is most pronounced on datasets with higher dimensionality and complex feature interactions, such as WDBC. On smaller or simpler datasets like Iris and Wine Quality, traditional gradient boosting methods (CatBoost, XGBoost) currently exhibit superior or comparable performance with potentially greater efficiency, suggesting that the benefits of NODE's deeper architecture are best realized when data complexity warrants it.

7 Recommendations for Future Research

The theoretical framework of NODE, while robust, opens several avenues for further investigation and extension:

1. **Advanced Differentiable Tree Structures:**
   - Explore beyond ODTs to incorporate differentiable versions of more expressive tree structures (e.g., non-oblivious trees, trees with oblique splits) within the NODE framework. This would involve developing differentiable mechanisms for more complex split criteria and feature combinations at nodes.
   - Investigate dynamic tree architectures where the depth or number of trees per layer could be learned or adapted during training.
2. **Refinement of the Entmax Mechanism:**
   - Conduct a deeper theoretical analysis of the role of the α parameter in α-entmax.
   - Explore alternative sparse, differentiable choice functions and compare their theoretical properties and impact on learning dynamics within the NODE context.

3. **Adaptive Complexity and Hyperparameter Optimization for Diverse Datasets:** Develop strategies to automatically adapt NODE's architecture (number of layers, trees per layer, or even the choice of entmax) based on different dataset characteristics (size, feature dimensionality, estimated complexity). This could improve NODE's performance across a wider range of tabular data, mitigating the observed performance drop on simpler datasets. This could involve meta-learning approaches or Bayesian optimization tailored to NODE's specific hyperparameter space.

References

[1] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[2] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning, 2016.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, and T. Y. Liu. Lightgbm. Advances in Neural Information Processing Systems 30 (NIPS), 2017.

[5] S. Popov, S. Morozov, and A. Babenko. Neural oblivious decision ensembles for deep learning on tabular data. International Conference on Learning Representations (ICLR), 2020.

[6] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. Advances in Neural Information Processing Systems 31 (NeurIPS), 2018.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations, 1986. Often cited as 1985 for the technical report version.

[8] Wikipedia contributors. Feature learning, 2025. Retrieved May 16, 2025, from https://en.wikipedia.org/wiki/Feature_learning#Hierarchical_feature_learning.