NeuralTree Models

A Differentiable Extension of Gradient Boosted Linear Trees

July 5, 2025

Introduction to NeuralTree

- Extension of Gradient Boosted Linear Trees (GBLT)
- Converts discrete (hard) splits into continuous (soft) splits
- Enables end-to-end training via backpropagation
- Initialized with pre-trained GBLT parameters, then fine-tuned
- Maintains interpretability through functional ANOVA decomposition
- Combines predictive power of tree ensembles with differentiability
- Mixture of experts (of linear models)

NeuralTree Model Architecture

Core Concepts

- Overall prediction is an ensemble of differentiable trees
- Each tree follows a depth-1 linear tree structure with soft splits
- Sigmoid function replaces hard decision boundaries
- Terminal nodes contain linear models
- Model is fully differentiable

Key Innovations

- Combines the interpretability of tree models with neural network differentiability
- Allows for gradient-based optimization of all model parameters
- Provides smooth decision boundaries while maintaining tree structure

Mathematical Foundation - Prediction Function

Overall Prediction Function

$$f(\mathbf{x}) = f_0 + \sum_{m=1}^{M} \gamma_m T_m(\mathbf{x}) \tag{1}$$

Components

- $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ - Feature vector
- f_0 - Baseline prediction (global mean)
- γ_m - Weight of the mth tree
- $T_m(\mathbf{x})$ - Differentiable tree function: mixture of linear models

Ensemble Structure

- Similar to traditional gradient boosting
- Each tree adds a refinement to the model
- Weighted sum of tree outputs
- Trees are differentiable, unlike standard trees

Tree Computation and Soft Splits

Soft Split Function

$$s_m(\mathbf{x}) = \sigma(a_m \cdot (x_{j_m} - t_m)) \tag{2}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function

Components

- x_{j_m} - Feature for splitting in tree m
- t_m - Threshold parameter
- a_m - Controls steepness of sigmoid (softness of split)

Hard vs. Soft Splits

- Hard split: Either left (0) or right (1) branch
- Soft split: Continuous value between 0 and 1
- When a_m is very large, approaches hard split
- When a_m is small, creates smoother transitions

Terminal Linear Models

Left Branch Linear Model

$$g_m^{(L)}(\mathbf{x}) = \beta_{m0}^{(L)} + \sum_{i=1}^{d} \beta_{mi}^{(L)} x_i \tag{3}$$

Right Branch Linear Model

$$g_m^{(R)}(\mathbf{x}) = \beta_{m0}^{(R)} + \sum_{i=1}^{d} \beta_{mi}^{(R)} x_i \tag{4}$$

Tree Output Equation

$$T_m(\mathbf{x}) = (1 - s_m(\mathbf{x})) \cdot g_m^{(L)}(\mathbf{x}) + s_m(\mathbf{x}) \cdot g_m^{(R)}(\mathbf{x}) \tag{5}$$

Interpretation

The output is a weighted average of two linear models, with weights determined by the soft split function $s_m(\mathbf{x})$

Two-Stage Training Process

1. Initialization with GBLT

- Train standard Gradient Boosted Linear Tree model
- Extract parameters: split feature, threshold, terminal model coefficients, tree weights
- Use these parameters to initialize NeuralTree

2. Refinement via Backpropagation

- With differentiable soft splits, entire model becomes end-to-end differentiable
- Optimize all parameters using gradient-based optimization:
  - Sigmoid steepness parameter a_m
  - Threshold t_m
  - Linear model coefficients
  - Tree weights γ_m
- Fine-tune for improved predictive performance and robustness

Using NeuralTree in MoDeVa Framework

```python

# For regression tasks
from modeva.models import MoNeuralTreeRegressor
model_neut = MoNeuralTreeRegressor(name="NeuralTree",
    n_estimators=100)

# For classification tasks
from modeva.models import MoNeuralTreeClassifier
model_neut = MoNeuralTreeClassifier(name="NeuralTree",
    n_estimators=100)
```

Training and Performance Assessment

```python
# Train model
model_neut.fit(ds.train_x, ds.train_y)

# Create TestSuite for evaluation
from modeva import TestSuite
ts = TestSuite(ds, model_neut)

# View model performance
result = ts.diagnose_accuracy_table()
result.table
```

TestSuite Features

- Comprehensive performance evaluation
- Multiple metrics for regression or classification
- Tools for model comparison and analysis
- Integrates with interpretation methods

Interpretability through Functional ANOVA

Decomposition of Prediction Function

The overall prediction function $f(\mathbf{x})$ is decomposed into additive components:

$$f(\mathbf{x}) = f_0 + \sum_i f_i(x_i) + \sum_{i<j} f_{ij}(x_i, x_j) + \dots \tag{6}$$

Baseline: Global mean prediction

$$f_0 \tag{7}$$

Main Effects: For each feature x_j

$$f_j(x_j) \tag{8}$$

Interaction Effects: For feature pairs (x_j, x_k)

$$f_{jk}(x_j, x_k) \tag{9}$$

NeuralTree Implementation Process

1. **Initialize with GBLT**
   Train GBLT model to get initial parameters
2. **Convert Hard Splits to Soft Splits**
   Replace hard thresholds with sigmoid function
   $s_m(\mathbf{x}) = \sigma(a_m \cdot (x_{j_m} - t_m))$
3. **Compute Neural Tree Output**
   $T_m(\mathbf{x}) = (1 - s_m(\mathbf{x})) \cdot g_m^{(L)}(\mathbf{x}) + s_m(\mathbf{x}) \cdot g_m^{(R)}(\mathbf{x})$
4. **Aggregate the Ensemble**
   $f(\mathbf{x}) = f_0 + \sum_{m=1}^{M} \gamma_m T_m(\mathbf{x})$
5. **Refine via Backpropagation**
   Optimize all parameters using gradient descent
6. **Apply Functional ANOVA**
   Decompose prediction function for interpretability

Global Model Understanding

Feature Importance

Impact of features on predictions

```python
result = ts.interpret_fi()
result.plot()
```

Importance Metrics

- Based on variance of marginal effects
- Normalized to sum to 1
- Higher values indicate stronger influence
- Accounts for feature scale differences

Effect Importance

Contribution of ANOVA components

```python
result = ts.interpret_ei()
result.plot()
```

Global Effect Plots

Visualize feature relationships

```python
# Main effect
result = ts.
    interpret_effects(
    features="hr")
result.plot()
```

Understanding Individual Predictions

Local Feature Importance

Feature impact for specific samples

```
result = ts.interpret_local_fi(
    sample_index=10,
    centered=True)
result.plot()
```

Local Effect Importance

Effect contribution for individuals

```
result = ts.interpret_local_ei(
    sample_index=10,
    centered=True)
result.plot()
```

Centering Options

Uncentered Analysis (centered=False):

- Raw feature contributions
- Direct interpretation
- May have identifiability issues

Centered Analysis (centered=True):

- Compares to population mean
- More stable interpretation
- Better for relative importance

Enforcing Domain Knowledge through Monotonicity

What is Monotonicity?

Ensures certain input features have consistently positive or negative effect on predictions

Examples of Monotonic Relationships

- In credit scoring, higher income $\rightarrow$ better credit ratings
- In medical risk assessment, increased risk factors $\rightarrow$ higher risk scores
- In pricing models, larger product quantities $\rightarrow$ higher total costs

Benefits of Constraints

- Aligns model with domain knowledge
- Makes model more reliable and trustworthy
- Prevents learning relationships that violate logical domain constraints
- Improves model behavior in sparse data regions

Implementing Monotonicity Constraints

Loss Function with Monotonicity

$$L(\theta) = l(\theta) + \gamma \sum_{i \in M} \max\left(0, -\frac{\partial \hat{y}}{\partial x_i}\right)^2 \tag{10}$$

Components

- $l(\theta)$ - Base prediction loss
- γ - Monotonicity penalty coefficient
- M - Set of features that should be monotonic
- $\frac{\partial \hat{y}}{\partial x_i}$ - Gradient of prediction w.r.t. feature i

Implementation in MoDeVa

- Specify monotonic features:
  - `mono_increasing_list=()`
  - `mono_decreasing_list=()`
- Control strength with `reg_mono`
- Start small, gradually increase
- Verify monotonicity holds for both main effects and interactions

Advantages of the NeuralTree Approach

Technical Benefits

- **Differentiability**
  End-to-end training capability
- **Improved Performance**
  Fine-tuning from GBLT initialization
- **Interpretability**
  Clear attribution through functional ANOVA

Practical Advantages

- **Smoothness**
  Soft splits create smoother decision boundaries
- **Domain Knowledge Integration**
  Through monotonicity constraints
- **Local and Global Explainability**
  Comprehensive interpretation tools

When to Use NeuralTree

Applications dealing with heterogeneous population through mixture of experts and interpretability while incorporating monotonicity constraints