GAMI-Net

Generalized Additive Models with Structured Interactions Network

July 5, 2025

Introduction to GAMI-Net

- Explainable neural network architecture
- Balances prediction accuracy with interpretability
- Extends traditional Generalized Additive Models (GAMs)
- Incorporates structured pairwise interactions
- Maintains interpretability through specific constraints
- Offers competitive predictive performance with high interpretability
- Handles both numerical and categorical variables
- Visualizes effects via 1D line plots and 2D heatmaps

Key Interpretability Constraints

Three Core Constraints

1. **Sparsity**: Selects only significant effects for a parsimonious model
2. **Heredity**: Includes interaction terms only if at least one parent main effect exists
3. **Marginal Clarity**: Distinguishes clearly between main effects and interactions

Implementation Approach

- Disentangled feedforward neural network
- Separate subnetworks for main effects and pairwise interactions
- Each feature has its own dedicated subnetwork
- Interaction subnetworks carefully selected and constrained

GAMI-Net Model Structure

Main Effects

- Individual subnetworks per feature
- Capture non-linear relationships
- Smooth, continuous shape functions
- One-dimensional effect visualization
- Can enforce monotonicity if needed

Pairwise Interactions

- Separate subnetworks for interactions
- Selected based on importance criteria
- Follow heredity constraints
- Two-dimensional effect visualization
- Enhance model flexibility

Advantages Over Tree-Based Models

- Continuous and smooth shape functions (more interpretable)
- Flexible incorporation of interpretability constraints

GAMI-Net Training Process

Stage 1: Train Main Effects

- Learn individual subnetworks for each feature
- Prune trivial main effect subnetworks
- Establish baseline for model performance

Stage 2: Model Pairwise Interactions

- Identify candidate interactions using heredity constraints
- Evaluate interaction importance using FAST scoring
- Train top-K, and prune trivial interaction subnetworks

Stage 3: Fine-Tune

- Retrain all components (main effects and interactions) simultaneously
- Apply regularization for interpretability
- Optimize overall model performance

Data Preparation for GAMI-Net

```
from modeva import DataSet
ds = DataSet()
ds.load(name="BikeSharing")

# Preprocessing steps
ds.scale_numerical(features=("cnt",), method="log1p")
ds.scale_numerical(features=ds.feature_names,
                   method="standardize")
ds.set_inactive_features(features=("yr", "season", "temp"))
ds.preprocess()

# Split data into training and testing sets
ds.set_random_split()
```

Important Preprocessing Notes

- Feature scaling is important for neural network subcomponents

Model Configuration in MoDeVa

```python
# For regression tasks
from modeva.models import MoGAMINetRegressor
model_GAMI = MoGAMINetRegressor()

# For classification tasks
from modeva.models import MoGAMINetClassifier
model_GAMI = MoGAMINetClassifier(max_epochs=(100, 100, 100))

# Train model
model_GAMI.fit(ds.train_x, ds.train_y.ravel())

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

Key Parameters

- max_epochs: Training epochs for each stage (main, interaction, joint)
- Additional hyperparameters available in MoDeVa documentation

Mathematical Foundation - Functional ANOVA

Basic Decomposition

The model decomposes into additive components:

$$f(\mathbf{x}) = \mu + \sum_j f_j(x_j) + \sum_{j<k} f_{jk}(x_j, x_k) \tag{1}$$

- $f_j(x_j)$ - Main effects for each feature
- $f_{jk}(x_j, x_k)$ - Pairwise interaction effects

Effect Computation

$$\text{Global average: } f_0 = \mathbb{E}[f(\mathbf{x})] \tag{2}$$

$$\text{Main: } f_i(x_i) = \mathbb{E}_{\mathbf{x}_{\setminus i}}[f(\mathbf{x}) \mid x_i] - f_0 \tag{3}$$

$$\text{Interaction: } f_{ij}(x_i, x_j) = \mathbb{E}_{\mathbf{x}_{\setminus \{i,j\}}}[f(\mathbf{x}) \mid x_i, x_j] - f_i(x_i) - f_j(x_j) - f_0 \tag{4}$$

Ensuring Interpretable Decomposition

Purification Constraints

- **Zero Mean Constraint**:
  $\int f_{i_1 \ldots i_t}(x_{i_1}, \ldots, x_{i_t})\, dx_k = 0, \quad k = i_1, \ldots, i_t$
- **Orthogonality Constraint**:
  $\int f_{i_1 \ldots i_u}(x_{i_1}, \ldots, x_{i_u}) \cdot f_{j_1 \ldots j_v}(x_{j_1}, \ldots, x_{j_v})\, d\mathbf{x} = 0,$
  for $(i_1 \ldots i_u) \neq (j_1 \ldots j_v)$

Effect Attribution

- Local Main Effect Contribution: $f_j(x_j)$
- Local Interaction Contribution: $f_{jk}(x_j, x_k)$
- Feature Attribution: $z_j(x_j) = f_j(x_j) + \frac{1}{2} \sum_k f_{jk}(x_j, x_k)$

Understanding Feature Impact

```
# Global feature importance
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

Interpretation Value

- Identifies key drivers of predictions
- Provides ranking of feature relevance
- Guides feature selection
- Supports model simplification

Understanding Component Contributions

```
# Global effect importance
result = ts.interpret_ei()
result.plot()
```

Importance Calculation

- Based on variance of individual ANOVA terms
- Shows main effects vs. interaction contributions
- Higher values indicate stronger influence
- Distinguishes between different model components

Categorical Variables

- One-hot encoded automatically
- Can view importance per category
- Interpretable through reference levels
- Visualized appropriately in plots

Visualizing Feature Relationships

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

# Interaction effect plot
result = ts.interpret_effects(features=("hr","workingday"))
result.plot()
```

Main Effect Plots

- 1D line plots
- X-axis: Feature values
- Y-axis: Effect on prediction
- Shows shape and direction of effect
- Reveals nonlinear patterns

Interaction Effect Plots

- 2D heatmaps
- Axes: Feature pair values
- Color: Joint effect magnitude
- Shows complex relationships
- Highlights regions of strong interaction

Individual Prediction Analysis

```
# Local feature importance
result = ts.interpret_local_fi(sample_index=10, centered=
    True)
result.plot()

# Local effect importance
result = ts.interpret_local_ei(sample_index=10, centered=
    True)
result.plot()
```

Visualization Components

- Feature or Effect contributions
- Feature values for specific sample
- Comparison to average behavior
- Direction and magnitude of effects

Centering Options

- **Uncentered:** centered=False
  - Raw feature contributions
  - Direct interpretation
- **Centered:** centered=True:
  - Compares to the true mean
  - Better for relative importance

Enforcing Domain Knowledge

Real-World Applications

- Credit scoring: Higher income $\rightarrow$ better credit ratings
- Medical risk assessment: Increased risk factors $\rightarrow$ higher risk scores
- Pricing models: Larger quantities $\rightarrow$ higher total costs

Why Monotonicity Matters

- Makes model more reliable and trustworthy
- Prevents learning relationships that violate logical domain constraints
- Particularly valuable in regions with sparse training data
- Aligns model with business knowledge and expectations
- Increases stakeholder confidence in model predictions

Implementing Monotonicity Constraints

Loss Function Components

$$L(\theta) = l(\theta) + \lambda \sum_{j \in S_1} \sum_{(j,k) \in S_2} \Omega(h_j, f_{jk}) + \gamma \sum_{i \in M} \max(0, -\frac{\partial \hat{y}}{\partial x_i})^2 \quad (5)$$

- $l(\theta)$ - Base prediction loss
- $\lambda \sum_{j \in S_1} \sum_{(j,k) \in S_2} \Omega(h_j, f_{jk})$ - Marginal clarity penalty
- $\gamma \sum_{i \in M} \max(0, -\frac{\partial \hat{y}}{\partial x_i})^2$ - Monotonicity penalty

Working Components

1. Prediction loss ensures accurate predictions
2. Marginal clarity term maintains separation between main effects and interactions
3. Monotonicity penalty penalizes negative gradients

Practical Implementation Tips

Configuration Parameters

- Specify monotonic variables:
  - `mono_increasing_list=()`
  - `mono_decreasing_list=()`
- Set regularization strength:
  - `reg_mono` parameter
- Control sampling:
  - `mono_sample_size` parameter

Tuning Strategy

- Start with small `reg_mono` value and gradually increase
- Evaluate both prediction performance and monotonicity violations
- Verify monotonicity holds for both main effects and interactions

Balancing Act

- Large γ: Strongly enforces monotonicity (may reduce accuracy)
- Small γ: More flexibility to fit data while maintaining some monotonic tendency
- Optimal setting depends on application and domain knowledge

Why Choose GAMI-Net?

Performance Benefits

- Competitive accuracy with black-box models
- Smooth, continuous shape functions
- Automatic feature selection
- Scales well to high-dimensional data

Interpretability Benefits

- Clear visualization of feature effects and structured interaction modeling
- Explicit domain knowledge integration
- Transparent prediction decomposition with both global and local explanations

Practical Advantages

- Balances accuracy-interpretability tradeoff
- Suitable for regulated industries requiring model transparency
- Facilitates communication with non-technical stakeholders
- Enables knowledge discovery from complex relationships