LEURN

A Generative and Explainable Neural Network for Tabular Data

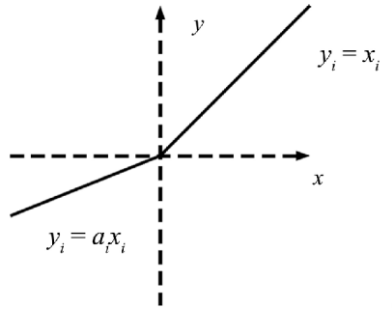
Caglar Aytekin

Outline

- Motivation
- LEURN
- Example

Motivation : Neural Networks are Decision Trees

- Neural Networks with piece-wise linear activations are:
  - Multivariate decision trees with factorized decision rules.
- 2-layer fully-connected neural network
  - $y = W_1^T \sigma(W_0^T x + \beta_0) + \beta_1$
  - Multivariate rule-set:
    - $w_{00}^T x + \beta_{00} > 0$, $w_{01}^T x + \beta_{01} > 0$, …
  - Factorization:
    - 2^F Possible effective matrices



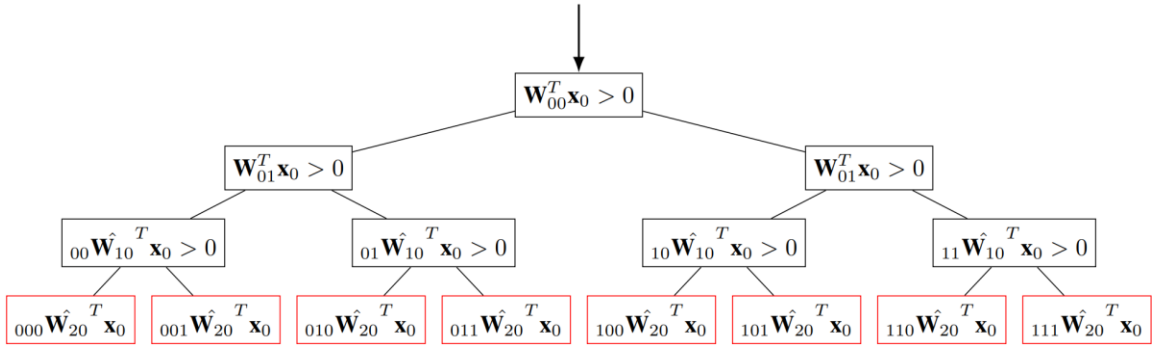


Figure 1. Decision Tree for a 2-layer ReLU Neural Network

https://arxiv.org/abs/2210.05189

Motivation

- Case: Electricity Demand
  - Factors: Temperature (T), humidity (H), wind (W), sunlight (L), season (S), precipitation (P), weekday (D), month (M), etc.
  - 5-layer neural networks will give you something like this:
    - $0.9T+0.7H-1.2W+0.5L+0.3S-0.8P+0.4D+0.6M+2.5>0$
    - $-0.5T+0.8H+1.1W+0.4L-0.9S+0.7P-0.3D+0.5M+3.0>0$
    - $1.2T-0.6H+0.7W-1.3L+0.9S+1.1P-0.4D+0.8M+1.0>0$
    - $-1.1T+0.9H+0.6W+1.4L+0.2S-0.5P+1.0D-0.7M+2.8>0$
    - $0.8T-1.2H+0.4W+0.7L+1.1S+0.9P+0.3D-0.6M+4.5>0$
    - If above holds, then there will be increase in demand
    - Can you explain this? Given these rules, can you seamlessly generate new data?

Motivation

- “Regular” Decision Trees can give you this:
  - $(0 < T < 1.2), (-0.5 < H < 2.5), (0.1 < W < 3.4), (1.0 < L < 4.5), (-2 < S < 0.8), (0.5 < P < 2.0), (-1.5 < D < 1.0), (0.2 < M < 2.8)$
  - Easy to explain, easy to generate new samples. But they are not as powerful as neural networks in terms of representational capacity.

- LEURN : Learning Univariate Rules by Neural Networks
  - Combines the representational power of “regular” neural networks and explainability and seamless data generation of “regular” decision trees.

LEURN

- Define Encoding Layer:
  - Takes NN input x and a vector called τ , returns: $sign(x - \tau)$
    - $e(x, \tau) = sign(x - \tau)$
    - Optional: apply dropout here
- Start by a directly learned τ : τ_0
- First encoding: $e_0 = e(x, \tau_0)$
- Next threshold: $\tau_1 = W_0^T e_0 + \beta_0$
- Next encoding and threshold: $e_1 = e(x, \tau_1), \tau_2 = W_1^T e_{0:1} + \beta_1$
- …
- Final layer: $\sigma(W_n^T e_{0:n} + \beta_n)$
  - σ : linear, sigmoid, softmax for regression, binary classification and multiclass classification respectively.

LEURN

- Univariate
  - Only non-linearity is at encodings:
    - $e(x, \tau) = sign(x - \tau)$
    - Partitioning (-1 or 1) operates separately on each feature.
- Exponential representation power
  - $\tau_1 = W_0^T e_0 + \beta_0$
  - Based on e_0, there are $2^{|e_0|}$ possible effective matrices and hence $2^{|e_0|}$ possible next thresholds

LEURN

- Explanations:
  - Categories
    - We store all thresholds : $\tau_{0:n}$
      - These form n number of thresholds for each input feature
      - Where input falls according to these thresholds are the only determining factor in NN output
      - Thus, we find lower and upper boundaries for each feature
  - Contributions
    - Final encoding $e_{0:n}$ is a binary (-1 or 1) signal
      - Every binary number here is directly associated with a feature (eg e_{00} comes from feature 0)
      - Notice that last layer is $\sigma(W_n^T e_{0:n} + \beta_n)$, so every feature has a distinct contribution in last result
      - These contributions are summed per feature and directly gives additive contribution

LEURN

- Generation
  - Generation from boundaries (eg data anonymization)
    - Simply calculate explanation (lower and upper boundaries per feature) for input sample x_0
    - Generate new sample $\overline{x_0}$ by randomly sampling from these lower and upper boundaries
    - As long as the category is same, neural network result is exactly the same
    - Hence, we can anonymize x_0 by $\overline{x_0}$
  - Generation from scratch
    - Randomly assign -1 or 1 to encodings in each layer
      - Not all paths are viable (one has to assign either 1 or -1), take this into consideration by monitoring upper and lower bounds per feature during layer-by-layer progression.
      - Example: In an early layer 0<x<1 was found, new threshold is 2, encoding has to be -1.
    - Find resulting upper and lower boundaries
    - Sample from those boundaries

LEURN

- LEURN-specific considerations:
  - Encoding Layer
    - For efficient learning in the backwards pass, we use derivatives of $tanh(x - \tau)$
    - For even further efficient learning, in first few epochs we use:
      - $(\alpha)sign(x - \tau) + (1 - \alpha)tanh(x - \tau)$
      - And gradually increase α to 1
        - Note: model is not monitored for saving until $\alpha = 1$, unless resulting model isn't a decision tree.
  - Normalizations
    - We normalize inputs so that they are in (-1,1) range
  - Initialization of weights
    - We know encodings will be binary (-1 or 1), and input range is (-1,1), hence we initialize W_i^T randomly (uniform) between $\left(-{}^{1}/_{|e_{0:n}|}, {}^{1}/_{|e_{0:n}|}\right)$, ensuring outputs are in (-1,1) range and serves as good thresholds to NN input (which is also in (-1,1) range)

LEURN

- General considerations:
  - Handling categorical values
    - All categorical values are label encoded
    - An embedding layer is learned for each category
    - Length of embedding layer is extracted according to formula: $\max(2, \log2(category\_number))$