Rule Extraction from Trained Neural Networks Using Regression Trees

Joy Chopra and Gaurav Nakum

April 2019

1 Introduction

Artificial Neural Networks (ANNs) are known to possess great generalization capabilities to unseen data in many problem domains. However ANNs are black boxes and cannot explain the reasons that were responsible for a decision (output). Rule extraction is one way of understanding the knowledge stored in a trained ANN.

Decision trees are considered the simplest and most interpret-able models in machine learning. This is because the predictions of a decision tree are simple *if ... then ...* rules. Also, they are fast in handling inputs of large dimensions and in making fast predictions. In this report, we present a method to extract rules from a neural network trained on regression problems using decision trees. Our approach to extract rules is not restricted to any specific NN architecture, but for simplicity we focus on fully connected networks for regression.

We plan to explain the predictions (post-hoc interpretability) of a NN by forming rules using decision tree induction and a pedagogical approach. Specifically, we would construct a decision tree for a given trained NN which makes predictions as close as those made by the NN. Although there have been many works on using decision trees for modeling classification networks, there are very few which tackle regression problems and thus this experiment.

2 Proposed Algorithm: TREPAN for Regression

We modify the TREPAN algorithm [2] for regression problem. To achieve this, we use the regression tree construction method of Brieman et. al. [1]. Specifically, to evaluate the split at each node, we check the MSE (mean-squared error) loss instead of Gini index as is used for classification. Also, the predicted regression value at each leaf node is the mean of the regression values of the training instances reaching that node during tree construction.

The TREPAN algorithm uses an oracle which is used to:

1. determine regression value for training examples
2. select splits for each internal node of the tree
3. determine if a node covers instances of only one class

Essentially, it achieves these tasks by generating samples from the training distribution of the network and making sure they obey the given constraints.

Algorithm 1 summarizes our proposed method.

3 Experiments and Results

3.1 Dataset

Our dataset contains 14 features - *Amplitude_Envelope, Amplitude_Weighted_Cosine_Phase, Amplitude_Weighted_Frequency, Amplitude_Weighted_Phase, Derivative, Dominant_Frequency, Instantaneous_Frequency, Instantaneous_Phase, Integrate, Integrated_Absolute_Amplitude, P-Impedance, Quadrature_Trace, Seismic, VpVs*

The task is to predict the variable - *PHIT* (Porosity)

The seismic data that is the features are recorded with respect to time whereas the output variable is recorded with respect to depth. They are then correlated using a preprocessing step called well tie.

The training data consists of 30005 instances while the test and validation datasets contained 3000 instances each.

Algorithm 1: TREPAN for Regression

Input: Oracle, Dataset - (X,Y)
Output: Regression Tree

Priority Queue Q := ϕ;
for *each* x *in* X **do**
    Regression value for x = Oracle(x)
Initialise the root of the tree as a leaf node;
Calculate priority for the root;
Put <root,X,{},$priority_{root}$ > in the Q;
while *Q is not empty and size(tree) <* $tree\_size\_limit$ **do**
    Remove node N from head of Q;
    $examples_N$: set of examples stored with N;
    $constraints_N$: set of constraints stored with N;
    Construct set of candidate splits P = { $x_1 \leq 0.1, x_1 \leq 0.2, ..., x_2 \leq 0.1, x_2 \leq 0.2..., x_{14} \leq 0.1...$ } ;
    Use $examples_N$ and calls to Oracle($constraints_N$) to evaluate the splits in P;
    Let S := best split ;
    Make N an internal node with split S;
    for *each outcome s of S:* **do**
        Make C, a new child node of N;
        $constraints_C := constraints_N \cup \{S{=}s\}$;
        Use call to Oracle($constraints_C$) to determine if C should remain a leaf ;
        if *C is not leaf* **then**
            $examples_C$:= members of $examples_N$ with outcome s on split S ;
            Calculate priority of C; Put <C,$examples_C$,$constraints_C$, $priority_C$ > into Q;

3.2 Preprocessing

The input data is min-max normalized:

$$x^i_{normalized} = \frac{x^i - MIN_{j=1,...,n} x^j}{MAX_{j=1,...,n} x^j - MIN_{j=1,...,n} x^j} \tag{1}$$

where x^i is a feature value $i = 1, ...14$.

3.3 Training the Network

We train a 4 hidden layer neural network with 100 units in each layer:
[14-100-100-100-100-1].

Each hidden layer is fully connected with ReLU activation. Xavier initialization [3] was used to initialize the weights and biases of the network. The network was trained for 1000 epochs with a learning rate of 0.0001 and batch size of 512. MSE loss was used to train the network.

Fig 2a shows the epoch-wise loss of the network. Fig 2b shows the batch-wise loss.

3.4 Evaluation

The extracted tree can be evaluated using multiple measures. For our current experiments, we evaluate the extracted tree using their fidelity measure.

For classification, fidelity is defined as the ratio of the number of instances for which the prediction of the tree matches the prediction of the neural network to the total number of instances (in the test data set). For regression, since there are no class labels, we need a measure of how similar the tree's predictions are to those of the neural network. There are multiple ways to do so. For the definitions of each of the fidelity measures below, we define $pred_{net}$ to be a vector of the predicted regression values obtained from the neural network on the test data and $pred_{tree}$ to be a vector of the predicted regression values obtained from the extracted decision tree on the test data.

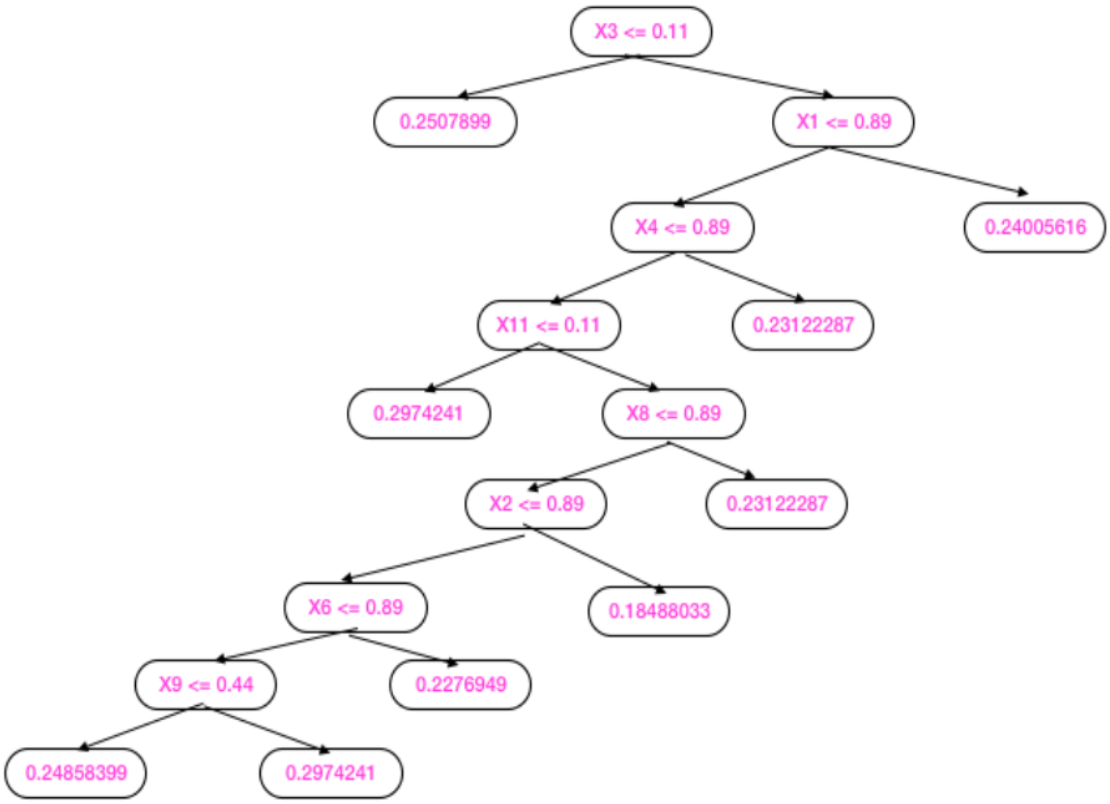
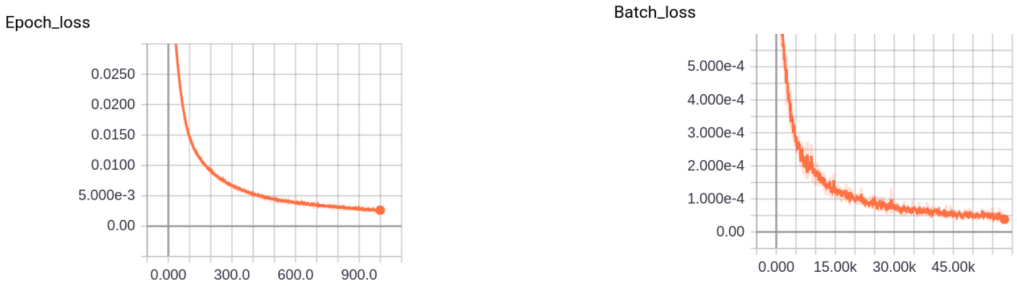


Figure 1: Extracted Decision Tree



(a) Epoch-wise loss

(b) Batch-wise loss

Figure 2: Loss graphs

No. of Nodes	EXP Fidelity	L1 Fidelity
3	**98.05** %	9.17 %
5	97.44 %	9.80 %
10	97.45 %	9.33 %
15	96.79 %	**12.63** %
20	96.33 %	11.67 %

1. L1/L2 Loss Fidelity: Let $loss_{thresh}$ be defined as a loss threshold. First we calculate the L1 loss vector between the two prediction vectors:

$$L1_{loss} = |pred_{net} - pred_{tree}| \tag{2}$$

   Then, L1 loss fidelity is defined as the ratio of the no. of elements in the vector $L1_{loss}$ having value less than or equal to $loss_{thresh}$ to the no. of test instances. The L2 loss fidelity is similarly defined.

2. Exp Loss Fidelity: This is similar to the L2 loss fidelity except that we map the L2 loss value (lying in range [0, ∞]) to the range [0, 1] using the exponential function exp(-x).

3. Cosine Similarity: The cosine similarity between the vectors $pred_{net}$ and $pred_{tree}$ is the dot product between these two vectors normalised by the product of their euclidean norms.

References

[1] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[2] Mark Craven and Jude W Shavlik. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems*, pages 24–30, 1996.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.