Analysis of Boundary Trees and Differentiable Boundary Trees

Jay Ricco, David Van Chu

August 8, 2017

Wentworth Institute of Technology

Introduction

The goal of this project was to understand, evaluate, and further study the properties of the Boundary Tree[1] and Differentiable Boundary Tree[2] algorithms, presented in their respective papers, *The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning*, and *Learning Deep Nearest Neighbor Representations Using Differentiable Boundary Trees*.

[1] https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/viewFile/9848/9953
[2] https://arxiv.org/abs/1702.08833

Boundary Trees

BT's - Introduction

- Learning Domain - Classification (Supervised)
- Goal - Fast to train, fast to query , on-line, instance based learning algorithm that can adapt to complex data distributions.
- Allows for data to self-distribute given distance and class.

BT's - Querying Procedure

```
begin BTQuery(x)
    Input: x - the query feature vector.
    Output: closest- closest node in the boundary tree.
    Initialize current to root node.
    while True do
        Let N be the set of child nodes of current.
        if |N| < k then
            N += {v}
        closest = arg min_{x_c ∈ N} d(x_c, x)
        if closest = current then
            break
        current ← closest
```

begin BTQuery($\mathbf{x}$)

- **Input:** $\mathbf{x}$ - the query feature vector.
- **Output:** closest- closest node in the boundary tree.
- Initialize current to root node.
- **while** *True* **do**
  - Let N be the set of child nodes of current.
  - **if** $|N| < k$ **then**
    - $N += \{v\}$
  - closest $= \arg\min_{\mathbf{x_c} \in N} d(\mathbf{x_c}, \mathbf{x})$
  - **if** closest $=$ current **then**
    - **break**
  - current $\leftarrow$ closest

BT's - Training Procedure

begin BTTrain($\mathbf{x}$, $\mathbf{y}$)

Input: $\mathbf{x}$ - the new examples feature vector.

Input: $\mathbf{y}$ - the new examples target class.

Initialize `closest` = BTQuery($\mathbf{x}$)

if `closest`.$y \neq \mathbf{y}$ **then**

Create node $\mathtt{v}_{new}$ in the Boundary Tree with position $\mathbf{x}$ and target class $\mathbf{y}$.

Add a new edge from `closest` $\rightarrow \mathtt{v}_{new}$.

BT's - Testing and Data

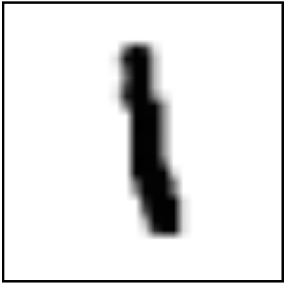
Our implementation of the algorithm was tested using three data sets:

- MNIST
- CIFAR-10
- Ground-Truth

Initial observations indicated that:

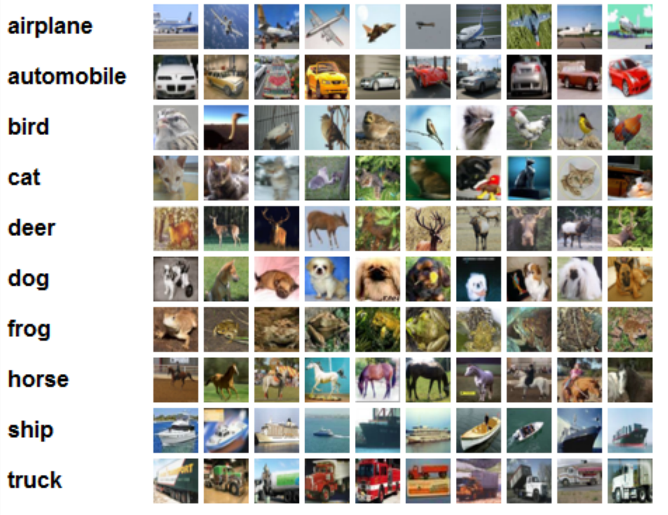
1. the max. branching factor value (k) is crucial with respect to the performance of the algorithm on a given data set.
2. While the total testing time and accuracy plateau, with $\uparrow$ examples, training time $\in \mathcal{O}(n)$.
3. BT's perform far better with MNIST than CIFAR; this shows inadequacy of the Euclidean distance metric for high feature complexity use-cases.

BT's - Testing and Data (MNIST Example)



$$\simeq \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

BT's - Testing and Data (CIFAR Example)



BT Observations - MNIST

k	Training Time (s)	Testing Time (s)	Accuracy (%)
2	75.96443768	14.89228232	79.248
3	76.26015964	14.11042054	82.253
5	72.54322867	14.19308667	84.988
10	93.82171679	18.15971713	87.397
20	128.2968537	25.59608793	88.211
50	165.1073234	34.94613609	88.036
100	164.9357249	35.71075509	87.952
∞	167.4871073	36.10234139	87.952

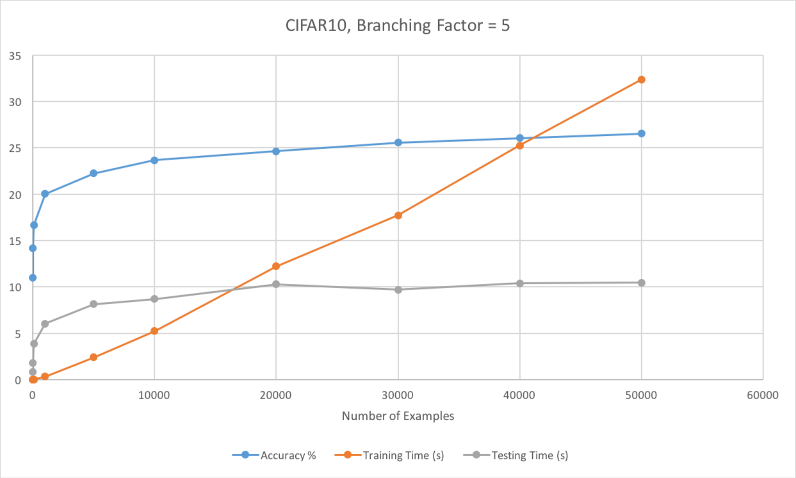
Table 1: For each test the branching factor is changed, and each data point was acquired by taking an average over 10 runs.

BT Observations - CIFAR-10

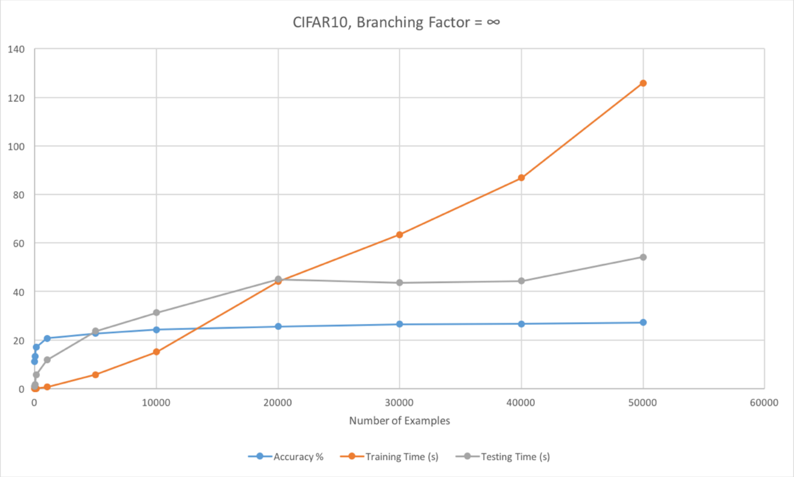
k	Training Time (s)	Testing Time (s)	Accuracy (%)
2	31.6216013	10.17820182	23.928
3	28.85782518	9.416226149	25.422
5	32.36283786	10.46060543	26.509
10	36.72829039	12.98402145	27.169
20	51.99976389	18.622286033	27.708
50	79.45825586	29.97079742	27.453
100	98.72666419	38.09738462	27.629
∞	125.9124599	54.22032864	27.283

Table 2: For each test the branching factor is changed, and each data point was acquired by taking an average over 10 runs. Notice how low the accuracy is.

BT Observations - CIFAR-10 (cont.)



BT Observations - CIFAR-10 (cont.)



Differentiable Boundary Trees

DBT's - Introduction

Key Difference: DBT's are augmented with a neural network that learns the *best* transform to apply on input features to maximize the L^2 distance between any two examples of different classes in the output space.
The algorithm dynamically builds neural networks which calculate the probabilities of an example being a specific class. This dynamically build network is then back-propagated through all the operations performed on it, distributing error to the proper parameters.

DBT's - Probabilistic Modeling of Traversals

- In order for any gradient-based machine learning method to work, there must be a hypothesis function, and it must be completely differentiable.
- Trees are inherently discrete, and we're traversing them layer-by-layer...
- How do we make our hypothesis differentiable?

DBT's - Probabilistic Modeling of Traversals

- Let's start by giving a firm definition to the distance function:

$$d(x_1, x_2) = \sqrt{\sum_k (x_{1,k} - x_{2,k})^2}$$

- With that defined, the probability of transitioning from a parent node to itself or any of it's children is:

$$p(\mathbf{x_i} \to \mathbf{x_j} | \mathbf{x_{query}}) = \text{SoftMax}_{i,j \in child(i)} \left(-d(\mathbf{x_j}, \mathbf{x_{query}})\right)$$
$$= \frac{\exp(-d(\mathbf{x_j}, \mathbf{x_{query}}))}{\sum_{j' \in (i, j \in child(i))} \exp(-d(\mathbf{x_{j'}}, \mathbf{x_{query}}))}$$

- Computing the full $P(path|\mathbf{x_{query}})$ distribution is hard; so we approximate the path through the tree, for some $\mathbf{x_{query}}$ by greedily selecting nodes at each level with the maximum calculated probability; this gives us the approximate path, $path*$.

DBT's - Final Class Probabilities

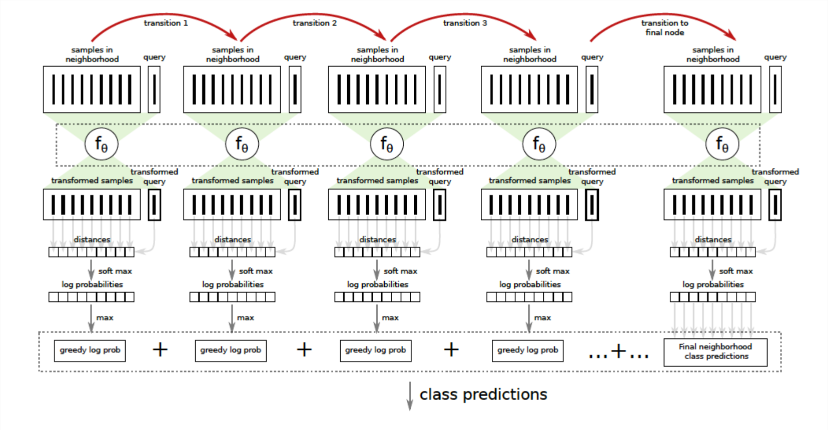
- The full equation to calculate the log class probabilities is as follows:

$$\log p(c|f_\theta(\mathbf{x_{query}})) = \sum_{\mathbf{x_i} \to \mathbf{x_j} \in path^* | \mathbf{x_{query}}} \log p(f_\theta(\mathbf{x_i}) \to f_\theta(\mathbf{x_j}) | f_\theta(\mathbf{x_{query}}))$$

$$+ \log \sum_{\mathbf{x_k} \in \text{sibling}(\mathbf{x_{final}})} p(\text{parent}(f_\theta(\mathbf{x_k})) \to f_\theta(\mathbf{x_k}) | f_\theta(\mathbf{x_{query}})) c(\mathbf{x_k})$$

- In other words, apply the transform to all the relevant feature vectors, then use the greedy path technique to calculate the log-class probabilities up until the last transition. At this point, all of the transitions to possible leaves are averaged over.

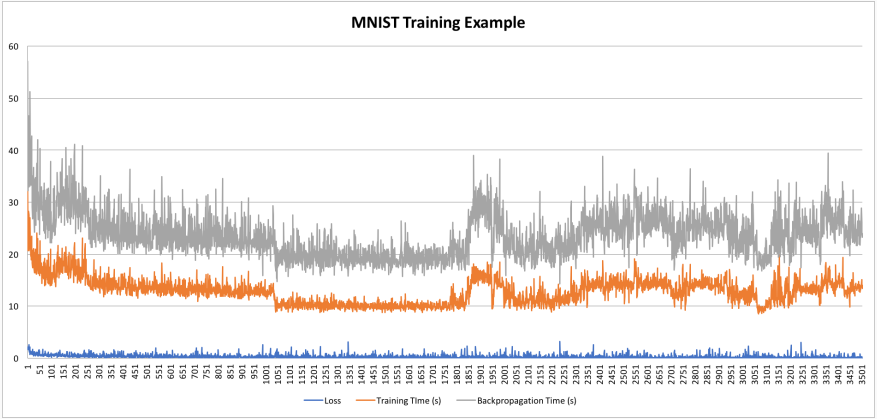
DBT's - Final Class Probabilities



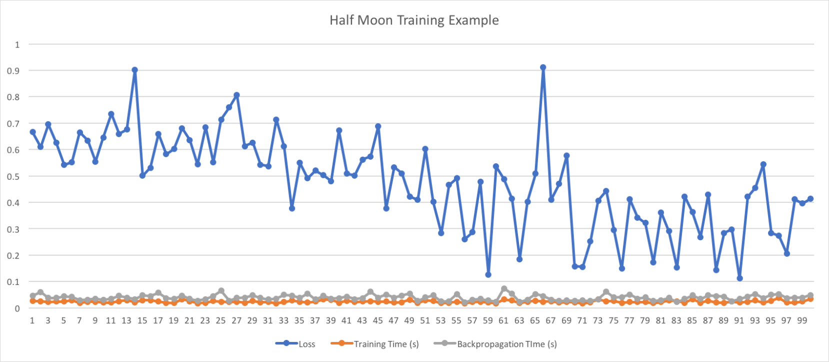
DBT's - Problems

- Dynamic network = Dynamic computation.
- The time it takes to perform a training batch is relatively constant, but the time to convergence is *large*. (For reference, one MNIST test took 30 hours!)
- Removes the whole point of having an on-line, fast, incremental learning algorithm.

DBT's - MNIST Data Set



DBT's - Half Moon Data Set



DBT's - Conclusions

- Ideal next steps would be to figure out a way to parallelize training.
- Otherwise, this method has limited usability due to time. (A standard neural network would do well enough.)
- Bogosort would be faster.