Introduction to Artificial Intelligence
Analysis of Boundary Trees and Differentiable Boundary Trees

Jay Ricco, David Van Chu

August 8, 2017

1 Motivation

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.

2 Boundary Trees

Features are the attributes of an input, or example. With the MNIST data set, each example is composed of 784 pixels, each one of which are considered the features.

The goal for Boundary Trees was to create a learning algorithm that is quick to train, quick to query, on-line and instance-based. On-line means that it's able to consistently accept and train new examples as it's running.

To query the Boundary Tree, we take an example as an input and find the closest node using Euclidean distance as the distance function. We return the closest node, and from that, we can get the class associated with the node. This is the class that the Boundary Tree thinks the example is.

To train the Boundary Tree, we query the existing tree with the example, and if the node returned does not have the same class as the example's target class, we add a new node to the tree with the example and corresponding target class.

3 Observations (MNIST)

Initial observations indicated that:

1. The maximum 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 $\epsilon O(n)$.
3. BT's perform far better with MNIST than CIFAR; this shows inadequacy of the Euclidean distance metric for high feature complexity use-cases.

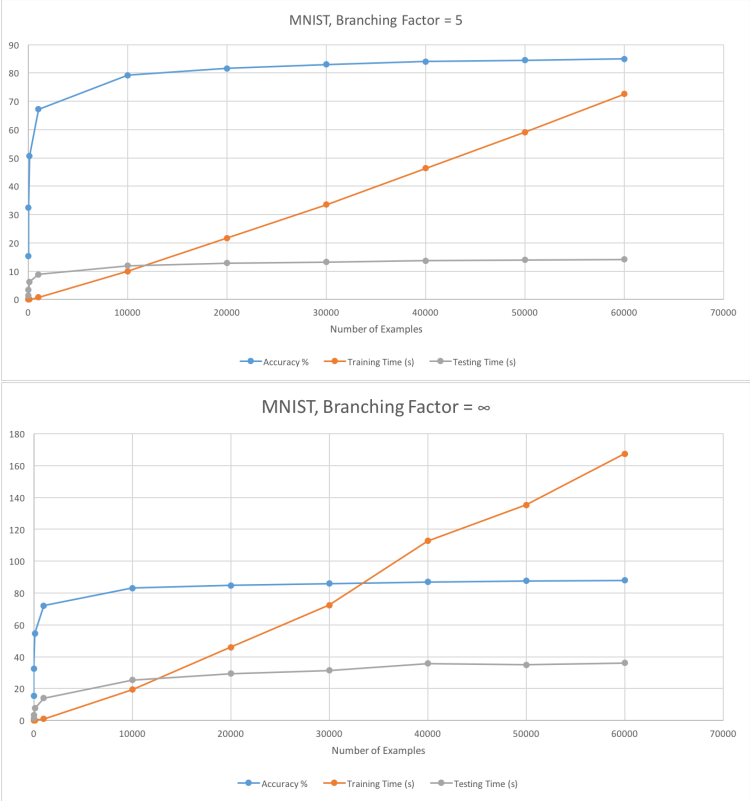
After running the tests, we noticed that setting the branching factor to 5-10 seemed to be the best if both time and accuracy is a concern, as anything less than 5 will result in lower accuracy while taking about the same amount of time to train and test. With branching factors larger than 10, training and testing time grew quite a bit with only a small increase in accuracy.

Another interesting point is that with a larger branching factor, the training time itself varies by quite a bit, ranging from 140.47 seconds to 223.79 seconds, whereas tests ran with a lower branching factor resulted in consistent training times.

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

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

Figure 2: MNIST dataset, varying the number of examples, averaged over 10 runs.



4 Observations (CIFAR10)

We see similar results when using Boundary Trees on the CIFAR10 dataset, although it never achieves over 27.7% accuracy. Using a smaller branching factor quickens the training time significantly compared to the MNIST dataset results.

Figure 3: For each test the branching factor is changed, and each data point was acquired by taking an average over 10 runs.

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

Figure 4: CIFAR10 dataset, varying the number of examples, averaged over 10 runs.