The Boundary Graph Supervised Learning Algorithm for Regression and Classification

Jonathan Yedidia

Disney Research
Outline

• Motivation
• Illustration using a toy classification problem
• Some simple refinements for regression and better generalization
• A high-dimensional problem (hand-written digit recognition using the MNIST database)
Context

• Boundary Graph Algorithm is a general-purpose supervised learning algorithm

• It’s a competitor to other general-purpose supervised learning algorithms like (deep) neural networks, support vector machines, K-nearest neighbor algorithms, or boosting
Desirable Features for Supervised Learning Algorithms

- Good generalization with few or many examples
- One-shot learning
- Can easily achieve zero error on training set
- On-line learning
- Fast processing of training examples and test queries
- Doesn’t require much memory for large training sets
- Query time grows very slowly with more training
- Able to absorb and learn from unlimited training examples
- Can learn and represent complex functions
- Easy to understand how and why it works
Boundary Graph Algorithm

• Satisfies all these features
• Caveat: requires that a metric be provided measuring distance between examples
• Applies to both classification and regression problems
• For demonstration purposes, we begin with a toy classification problem where inputs are points in a 2D square, and outputs are one of 8 colors. Distance metric is ordinary Euclidean distance.
Boundary Graph

• Nodes of the graph represent labeled training examples

• Edges exist between nodes that have similar inputs and dissimilar outputs

This boundary graph has a single “entry node”
To query a single-entry-node-boundary graph start at the entry node, and keep greedily moving to the neighboring node that is closest to the test query. Return the label of the locally closest node.

Colors for background non-example pixels are obtained using this procedure in this image.
To train a boundary graph on a new example, follow the same procedure (start at the entry node, and walk to the node locally closest to the example) and if the locally closest node has a different label from the example, add the example to the graph with a link to the locally closest node.

One-shot learning property is obvious.

(34th training example; 17th node in the BG)
After 10,000 training examples, the Boundary Graph has 724 nodes
We can drive the training error to zero and slightly improve generalization by running through the training examples a couple more times (now BG has 840 nodes)
Demo for toy classification problem

Ground truth
Using 500,000 training examples, the Boundary Graph with 6587 nodes (obtained in a few seconds) has nearly no test error.
Desirable Features for Supervised Learning Algorithms

- Good generalization with few or many examples
- One-shot learning
- Can easily achieve zero error on training set
- On-line learning
- Fast processing of training examples and test queries
- Doesn’t require much memory
- Query time grows very slowly with more training
- Able to absorb and learn from unlimited training examples
- Can learn and represent complex functions
- Easy to understand how and why it works
Comparison with Nearest-Neighbor Algorithm

Nearest Neighbors: 10,000 nodes

BG: 840 nodes from 10,000 examples
Comparison with Nearest-Neighbor Algorithms

- Both Nearest Neighbors (NN) and BG algorithm have zero training error.
- For test queries, the BG algorithm only returns the result of a training example that is locally nearest, not necessarily the globally nearest.
- Thus, it's not surprising that for the same number of training samples, NN typically has slightly better generalization than the simple BG I've described (e.g. 1.95% error versus 2.41% error with 10,000 samples). [This can be fixed using refinements to be explained]
- NN uses much more memory to store examples.
- BG has a query time that only grows logarithmically with the number of stored examples, so very slowly with number of training examples. NN query time will grow rapidly (linearly) with the number of training examples unless something clever is done.
- To my knowledge, other clever algorithms like KD-trees or locality-sensitive hashing have problems compared with BG's (not working well for large dimensionality for KD-trees, or only giving small speed-ups compared to boundary graphs for LSH)
Boundary Graphs for Regression

• The output for each example will now be some scalar or vector.

• We need some notion that the output of a locally nearest node is “similar” or “dissimilar” from the output of the training example. Typically just use some threshold distance in output space.

• If we don’t refine the algorithm, the reconstructed function will have a blocky step-function form, at least if the number of training examples is small.
Potential problem: blocky reconstruction
Smoothing the reconstruction

1. We can use multiple entry nodes in a single boundary graph, and average over the locally closest nodes obtained starting at different entry nodes.

2. Instead or in addition, we can average over “committees” of different boundary graphs. This option is simpler and also usually gives better generalization.

Both approaches improve the generalization performance for both regression and classification problems so that it is approximately the same or better than nearest-neighbor algorithms given the same number of training examples.

The trade-off is that we need to do more processing per training or test sample, either to process walks from multiple entry nodes, or to process multiple boundary graphs in a committee. Nevertheless, these approaches will be much faster than Nearest Neighbors if we choose the number of entry nodes or number of committee members reasonably.

But please remember that if a lot of training examples are available, Boundary Graphs let us use them much more efficiently than Nearest Neighbors—lots of data improves the generalization much more significantly than these refinements.
How do we even smooth ordinary nearest neighbors for regression?

We can use K-NN, and weight the nearest neighbors by a kernel function like the inverse distance.

Using Weighted 5-NN
BG’s with multiple entry nodes

• The first $E$ training examples will be entry nodes.

• Further refinements: only make $K$ walks for test queries, from the $K$ entry nodes closest to the test example, and similarly only make $L$ walks for training examples, from the $L$ entry nodes closest to the training example.

• At test time, make a weighted average of the set of locally closest nodes found.

• At training time, add a link from a locally closest node if it has a dissimilar output to the training example, as usual.

• Efficiency trick: cache results of a walk from a particular node.

• Note that the K-NN algorithm is actually a special case of our general BG algorithm, when $E=N$ (all nodes are entry nodes), and $L=1$. 
A Boundary Graph with 10 entry nodes

(Created using $E=L=K=10$)
Making Committees

- If we are OK with dealing with the training data in batch mode, we can just create different boundary graphs by shuffling the order in which the training data is presented. This approach is the most effective for improving generalization (test) performance.

- Otherwise if maintaining online processing is important, we can create a different boundary graph by just beginning with a slightly later training sample, which gives us a different set of entry node(s). This approach is good enough to smooth reconstruction and improves generalization to nearest-neighbor levels.

- For test queries, simply equally weighting the outputs of the different boundary graphs in the committee works well.
Typical results (1000 training samples)

5-NN
Error = 6.13%

One BG
Error = 7.92%

Online Committee of Twenty BG’s
Error = 6.29%

But don’t forget that the improvement from being able to add more training samples is much more significant.
Demo for toy regression problem

Ground truth

BG after 10,000 samples (395 nodes)
Higher Dimensional Problems

- Executive Summary: Boundary Graph Algorithm works well if you can find a reasonable distance metric.
- MNIST: Handwritten digit recognition. 28x28 grey-scale images with their labels. Training Set of 60,000 examples, Test Set of 10,000 examples.
- I used a simple Histogram-of-Gradients (HOG) representation, converted into a 2916-dimensional vector.
- My HOG representation had 4 possible gradients at each pixel (horizontal, vertical, and two diagonals), and histograms were created using a pixel and its 8 nearest neighbors.
Original Image

Histogram of Gradients Representation

27x27x4=2916 dimensional
Some Statistics

- All statistics are for a single-core Java program on my Macbook Pro laptop. BG algorithms fit in less than 2GB of RAM, most of which was used to store HOG vectors.

- Nearest-Neighbor Algorithm: 98.6 - 98.7% accuracy on test set. It processes about 0.5 queries/sec; thus 5 hours or so for the 10000 queries in the test set.

- Single Boundary Graph with one entry node: 94.1 - 94.6 % accuracy, but processes about 1000 queries/sec, thus 10 seconds for the 10000 queries in the test set, and about a minute for a pass through the 60000 examples in the training set (5 passes required to obtain zero training error).

- Batch-mode committee of 25 boundary graphs with one entry node: (different BG’s obtained by shuffling the order of the training set): 98.9-99.0% accuracy. With serial processing, takes 25 times as long as single boundary graph, but can easily be parallelized.
Summary

- Good generalization with few or many examples
- One-shot learning
- Can easily achieve zero error on training set
- On-line learning
- Fast processing of training examples and test queries
- Doesn’t require much memory for large training sets
- Query time grows very slowly with more training
- Able to absorb and learn from unlimited training examples
- Can learn and represent complex functions
- Easy to understand how and why it works
- Should work with any problem where you can define a reasonable metric.
- May want to use committees of boundary graphs to obtain best generalization.
Big Picture

• Like nearest-neighbor algorithms, boundary graphs let you have fast, online (and one-shot) learning of functions that have arbitrarily complicated form.

• Like neural networks, SVM’s, or other parametric algorithms, boundary graphs let you get fast responses to your queries and use reasonable amounts of memory even after training on lots of data.