Loopy Belief Propagation on MRFs in ITK

Release 0.00

David Doria

March 1, 2011

Rensselaer Polytechnic Institute, Troy NY

Abstract

This code provides a base implementation of Loopy Belief Propagation on MRFs in ITK. We use binary image denoising as an example problem to demonstrate this code.

Contents

1 Introduction

This code provides a base implementation of Loopy Belief Propagation on MRFs in ITK. We use binary image denoising as an example problem to demonstrate this code.

This document is intended only to describe the implementation, not the theory. A complete tutorial on loopy belief propagation on MRFs is available here:

https://github.com/daviddoria/Tutorials/blob/master/BeliefPropagation/BeliefPropagation.pdf?raw=true

2 Message Update Rules

We have implemented the sum-product, max-product, and min-sum message update rules. They are all very similar. The sum-product rule is typically used in Loopy BP. It finds the best label for each node individually, minimizing the number of nodes with incorrect labels. The max-product message update rule finds the image labeling with the maximum posterior probability. The min-sum update rule is identical to the max-product update rule, but the computations are performed in the negative log domain.

3 Message Update Schedules

There is no "best" schedule to update message in loopy belief propagation. We provide two sample schedules. The first, RandomUniqueUpdateSchedule, selects a node at random and updates the outgoing messages to all of its neighbors. The nodes are chosen without replacement until all nodes have been updated, at which point the process can be repeated. The second, RasterOneNeighborUpdateSchedule, performs a raster scan of the image nodes and updates the same selected neighbor for each node. That is, the first node is selected and its left neighbor updated, then the next node (raster scan order) is selected and its left neighbor updated, etc. Once every node has been visited, the raster scan starts from the beginning, this time updating the right node of each neighbor. For all of the messages to be updated, four raster scans must be performed. We tried a schedule which made a single raster scan, updating all neighbors of every node visited simultaneously, but this performed very poorly - the information from the corner of the image was propagated much too strongly, sometimes resulting in output images with almost all of the nodes taking the same label.

4 Demonstration: Binary Denoising

To demonstrate our Loopy BP algorithm we have chosen the binary denoising problem. This problem takes a binary image as input and outputs a binary image that has had the "noise removed". To solve this problem using the BP framework presented, we must simply specify an appropriate label set and cost functions.

4.1 Label set

In this problem, the label set L is $\{0,1\}$, and hence $|L| = 2$. That is, every node (pixel) can either take the value 0 or 1 (a binary image).

4.2 Unary Cost

In this problem, the input image is called the "observations". The cost of assigning a node a particular label is related to if the label agrees with the observations. One such function is:

$$Unary(node, label) = \begin{cases} .2 & \text{if } observation(node) = label \\ .8 & \text{otherwise} \end{cases} \quad (1)$$

This function encourages the resulting labeling to be the same as the observations. If a node's label is the same as its original observation, the cost is .2. If a node's label is different from its original observation, the cost is .8.

4.3 Binary Cost

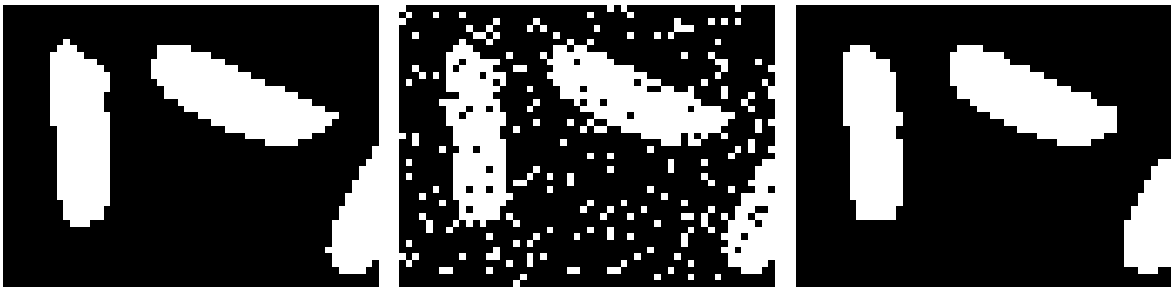
In a denoising problem, typically "smoothness" is the goal. That is, a node is likely to take the same label as its neighbors. The binary cost should encourage this smoothness. One such function to achieve this is:

$$Binary(label1, label2) = abs(label1 - label2) \quad (2)$$

According to this cost function, if a node's label is the same as its neighbor, the cost is 0. If neighboring labels are different, the cost is the difference between the labels.

4.4 Demonstrations

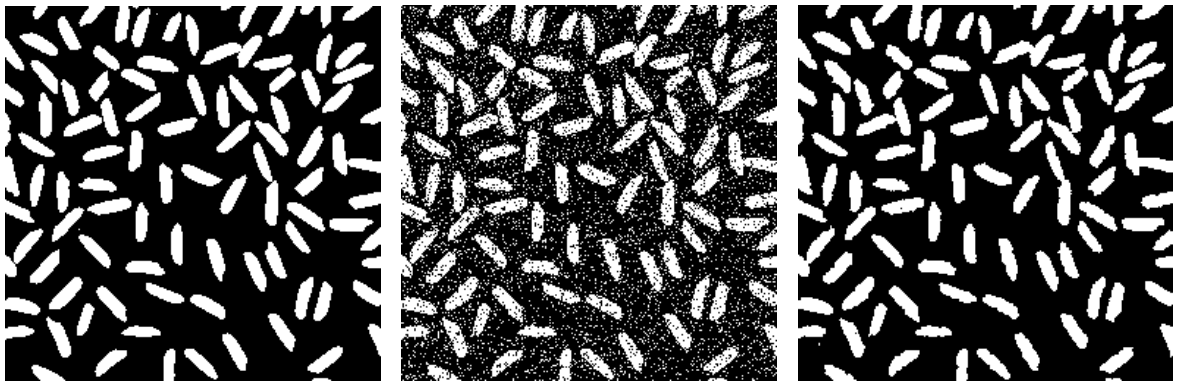
This small 56x42 example takes less than one second to run. This is the result of two passes with the Sum-Product update rule and the RandomUniqueUpdateSchedule.



(a) Original image (b) Image with noise added (c) Denoised image

Figure 1: A demonstration of binary denoising

Two iterations on this 256x256 image took approximately 10 seconds.



(a) Original image (b) Image with noise added (c) Denoised image

Figure 2: A demonstration of binary denoising on a larger image

5 Code Structure

- LoopyBP class

  This is an abstract base class to provide the interface for a loopy belief propagation algorithm. The functions UnaryCost and BinaryCost must be implemented by a subclass. LoopyBP performs the actual message passing algorithm.

- BinaryDenoising class

  This is a subclass of LoopyBP. It contains the implementations of the cost functions specific to the binary denoising problem.

- TestBinaryDenoisingReal.cxx

  This is the driver for the binary denoising demonstration. It accepts a noisy binary input image and produces a denoised output image.

- Node class

  This class represents a node in the MRF. It contains all of the nodes outgoing messages, as well as its position in the grid.

- Message class

  The Message class contains the label that the message is "talking" about and the value of the message.

- MessageVector class

  When two nodes talk to each other, the actually send a vector of messages, one for each label. This class encapsulates this vector of Messages.

- UpdateSchedule class

  This is an abstract base class for scheduling algorithms. It's core function is to provide the next MessageVector to process. The LoopyBP class contains an object of an UpdateSchedule subclass.

- RasterOneNeighborUpdateSchedule class

  This scheduling algorithm traverses the nodes of the MRF passing all of the messages to one of the current node's neighbors along the way. Four raster scans must be performed to update all messages.

- RandomUniqueUpdateSchedule class

  This scheduling algorithm selects a node at random (without replacement) and passes all of the outgoing messages to all of its neighbors. Once all nodes have been visited, the procedure starts over.

6 Code Snippet

Using this class only involves specifying a few parameters as shown below:

```
BinaryDenoising binaryDenoising;
binaryDenoising.SetObservations(binaryImage);
binaryDenoising.CreateBinaryLabelSet();

binaryDenoising.SetScheduleToRandomUnique();

binaryDenoising.SetUpdateToSumProduct();
binaryDenoising.SetBinaryPenalty(1.);
binaryDenoising.CreateAndInitializeMessages(1.0);
binaryDenoising.SetNumberOfPasses(2);
binaryDenoising.Initialize();

while(!binaryDenoising.IsFinished())
  {
  binaryDenoising.Iterate();
  }

Helpers::WriteBinaryImage<IntImageType>(
  binaryDenoising.GetResult(), "result.png");
```