MAP inference

CS B553
Spring 2013

Announcements

- A3 released soon!
- Quiz coming soon?

Loopy BP

- Run sum-product belief propagation on a cluster graph, instead of a clique tree
- Loopy BP can be **much** faster than exact inference
  - Running time?
- Loopy BP is not guaranteed to find the best solution
- It's not even guaranteed to converge to a solution
  - However, in practice it seems to converge to a reasonably good solution for a variety of Markov net problems

$$\delta_{i\to j}(\mathbf{S_{i,j}}) = \sum_{\mathbf{C_i}-\mathbf{C_j}} \psi_i(\mathbf{C_i}) \prod_{k\in\mathcal{N}(i)-\{j\}} \delta_{k\to i}(\mathbf{S_{k,i}})$$

Loopy BP on grid graphs

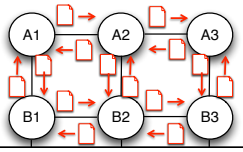
- In the special case of grid graphs, we can run BP on the Markov net directly, instead of the cluster graph
- This is possible because we can create a cluster graph with a very simple structure
- The computation is exactly the same, but it's more convenient to understand and implement this way

Belief propagation

- Messages are passed between the variables
  - Message from node *i* to *j* at iteration *t* is,

$$m_{i\to j}^{(t)}(X_j) = \sum_{X_i} \phi(X_i, X_j) \prod_{k\in\mathcal{N}(X_i)-\{X_j\}} m_{k\to i}^{(t-1)}(X_i)$$

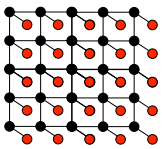
  - Intuitively (and anthropomorphically) a message from me to you says, for each of your states, "If you choose state Xj, here's how happy my neighbors and I would be."



Application: image reconstruction

- Images consist of arrays of pixels, each having a grayscale intensity value
- Given a noisy image N, estimate original image O
  - The O we choose should be similar to N
  - O should be "smooth"; neighboring pixels should be similar

Observed, noisy pixels



Original, unobservable pixels (in black)

- Need two kinds of clique potentials
  - Potentials between N and O variables
  - Potentials between O variables
  - Could encode using tables, but a simpler form makes sense

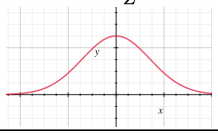
Image reconstruction potentials

- Potential between N and O pixels, $\phi(O_{i,j}, N_{i,j})$
  - Should be high when Oi,j and Ni,j are similar, low when they are very different
  - One possibility: use a function of their difference,

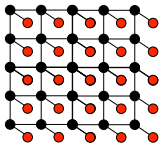
$$\phi(O_{i,j}, N_{i,j}) = f(O_{i,j} - N_{i,j})$$

  - For f(), a zero-mean Normal distribution,

$$f(x) = \frac{1}{Z} e^{\frac{-x^2}{2\sigma^2}}$$



Observed, noisy pixels



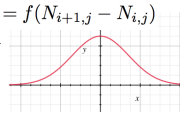
Original, unobservable pixels (in black)

Image reconstruction potentials

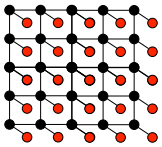
- Potential between neighbors in N, $\phi(N_{i,j}, N_{i+1,j})$ and $\phi(N_{i,j}, N_{i,j+1})$
  - Can use a similar function:

$$\phi(N_{i,j}, N_{i+1,j}) = f(N_{i+1,j} - N_{i,j})$$

$$f(x) = \frac{1}{Z} e^{\frac{-x^2}{2\sigma^2}}$$

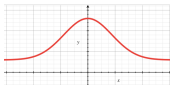


Observed, noisy pixels



Original, unobservable pixels (in black)

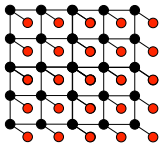
  - In practice, we don't want to penalize discontinuities so much; e.g. instead use:



Inference

- Use Loopy Belief Propagation to perform inference
  - i.e. estimate $P(O_{i,j} \mid N)$, for all pixels $O_{i,j}$
  - Do this by exchanging messages between black nodes until convergence (or for some maximum # of iterations)

Observed, noisy pixels



Original, unobservable pixels (in black)

Corrupted

Restoration

[FH06]

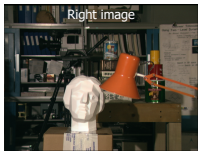
Another application: Stereo

- Given images of the same scene taken from two different perspectives, estimate depth of each pixel

Kim Scarborough

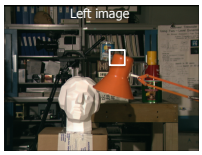
Solving stereo

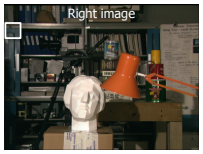
- For every pixel in left image, we want to find the corresponding pixel in the right image
  - Amount of "movement" or *disparity* tells us the depth



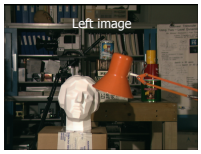
Block Based Matching

- How to determine these correspondences?
  - A simple method is to take a small patch from the left image, and "move" it around the right image until we find a good match

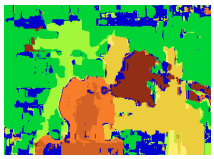




Results with window search

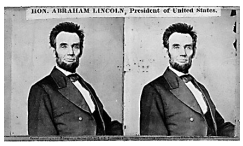


Block-based matching



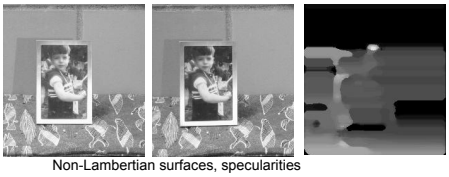
Ground truth

Failures of correspondence search



Textureless surfaces

Occlusions, repetition

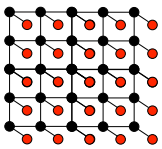


Non-Lambertian surfaces, specularities

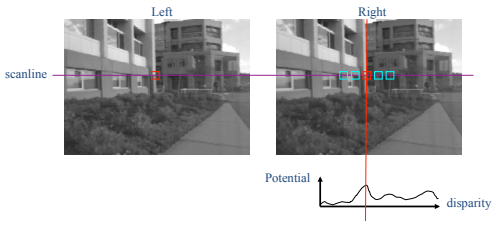
Markov network formulation

- For each pixel of the left image, we want to find a corresponding right pixel such that:
  - The image data around the 2 pixels match well
  - The resulting disparity estimates are "smooth"
- What will potential functions be?

Observed pixel data



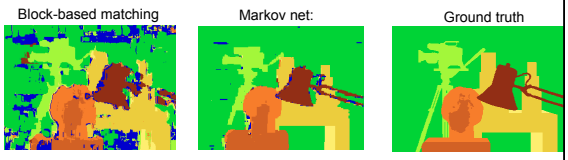
Unobservable disparities



- Potential function between observed and unobserved variables given by block correspondences, and will be different for each pixel

Approximate MRF inference

- These algorithms are not well understood
  - But they seem to work well in practice



Making inference tractable

- In practice, making inference tractable is a key challenge in applying graphical models to applications
- Typically, the options are:
  - **Exact inference** with **arbitrary potentials** on a graphical model, but with a **simplified structure**
  - **Exact inference** on a graphical model with **arbitrary structure**, but **restricted potentials**
  - Graphical model with **arbitrary structure** and **arbitrary potentials,** but settle for **approximate inference**

Types of inference

- Given set of observed variables **X**, and sets of unobserved variables **Y** and **W**…

Marginal query: What's the distribution of P(**Y**|**Z**)?	**MAP query:** What are the most likely values of **Y** and **W** given **X**?	**Marginal MAP query:** What are the most likely values of **Y** given **X**?
$P(\mathbf{Y}\|\mathbf{X}) = \frac{1}{Z}\sum_{\mathbf{W}} P(\mathbf{X},\mathbf{Y},\mathbf{W})$ $= \frac{1}{Z}\sum_{\mathbf{W}}\prod\phi(\cdot)$	$\mathbf{Y}^*,\mathbf{W}^* = \arg\max_{\mathbf{Y},\mathbf{W}} P(\mathbf{X},\mathbf{Y},\mathbf{W})$ $= \arg\max_{\mathbf{Y},\mathbf{W}}\prod\phi(\cdot)$	$\mathbf{Y}^* = \arg\max_{\mathbf{Y}} P(\mathbf{X},\mathbf{Y},\mathbf{W})$ $= \arg\max_{\mathbf{Y}}\sum_{W}\prod\phi(\cdot)$

MAP inference

- In Maximum A Posteriori (MAP) inference, the goal is to simultaneously find the most likely values for *all other variables* given some observed variables

$$\mathbf{Y}^*,\mathbf{W}^* = \arg\max_{\mathbf{Y},\mathbf{W}} P(\mathbf{X},\mathbf{Y},\mathbf{W})$$

$$= \arg\max_{\mathbf{Y},\mathbf{W}}\prod\phi(\cdot)$$

  - Note that this is different from computing a marginal over each variable and then maximizing them separately (as in Assignment 2)

MAP inference

- We can perform MAP inference using a simple modification to the Variable Elimination algorithm

Variable elimination on Markov nets

1. Sort the non-query variables in an arbitrary order, Z1, Z2, … Zn
2. Initialize set of *factors* **F** to be the factors in the Markov net, ϕ_1, ϕ_2, …
3. For each i=1..n,
   a. Identify subset of factors **F'** involving Zi; these factors have some subset of variables **V** as parameters
   b. Take product of factors **F'**, producing ψ(**V**)
   c. Sum this product over all values of Zi, producing a new factor τ parameterized by **V**-{Zi}
   d. Remove **F'** from **F**, then add τ to **F**

Variable elimination on Markov nets

1. Sort the non-query variables in an arbitrary order, Z1, Z2, … Zn
2. Initialize set of *factors* **F** to be the factors in the Markov net, ϕ_1, ϕ_2, …
3. For each i=1..n,
   a. Identify subset of factors **F'** involving Zi; these factors have some subset of variables **V** as parameters
   b. Take product of factors **F'**, producing ψ(**V**)
   c. ~~Sum~~ Maximize this product over all values of Zi, producing a new factor τ parameterized by **V**-{Zi}
   d. Remove **F'** from **F**, then add τ to **F**

VE for MAP queries

- Result of this modified VE algorithm is a single factor containing the probability of the most-likely assignment of variables, i.e. $\max_{\mathbf{Y}} P(\mathbf{X}, \mathbf{Y})$
- All of the results we derived for the original VE algorithm apply here as well, e.g.:
  - Order in which you eliminate variables doesn't matter to correctness (but might change running time)
  - Finding fastest VE ordering is NP hard
  - Running time depends on tree width of graph
  - Can take exponential time in the worst case

Finding the MAP solution

- Finding the actual assignment (i.e. $\arg\max_{\mathbf{Y}} P(\mathbf{X}, \mathbf{Y})$) is easy given the intermediate factors of VE
- Start with the last factor and "trace backwards"
  - Choose values that maximize the last factor over its scope
  - Then choose values that maximize next-to-last factor over its scope, consistent with values you've already chosen
  - Continue backwards until reaching the first factor

Max-product algorithm

- We can also define a message-passing algorithm to help compute MAP solutions
- Recall the Sum-product algorithm, which passes messages over nodes of a clique tree:

$$\delta_{i \to j}(\mathbf{S_{i,j}}) = \sum_{\mathbf{C_i - C_j}} \psi_i(\mathbf{C_i}) \prod_{k \in \mathcal{N}(i) - \{j\}} \delta_{k \to i}(\mathbf{S_{k,i}})$$

- The Max-product algorithm is very similar,

$$\delta_{i \to j}(\mathbf{S_{i,j}}) = \max_{\mathbf{C_i - C_j}} \psi_i(\mathbf{C_i}) \prod_{k \in \mathcal{N}(i) - \{j\}} \delta_{k \to i}(\mathbf{S_{k,i}})$$

Finding the MAP solution

- Recall that with Sum product BP, marginals over cliques were computed from incoming messages,

$$P(\mathbf{C_i}) = \frac{1}{Z} \psi_i(\mathbf{C_i}) \prod_{k \in \mathcal{N}(i)} \delta_{k \to i}(\mathbf{S_{k,i}})$$

- For Max product BP,

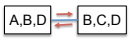
$$\mathbf{C_i^*} = \max_{\mathbf{C_i^*}} \psi_i(\mathbf{C_i}) \prod_{k \in \mathcal{N}(i)} \delta_{k \to i}(\mathbf{S_{k,i}})$$

Loopy max-product BP

$$\delta_{i\to j}(\mathbf{S_{i,j}}) = \max_{\mathbf{C_i}-\mathbf{C_j}} \psi_i(\mathbf{C_i}) \prod_{k\in\mathcal{N}(i)-\{j\}} \delta_{k\to i}(\mathbf{S_{k,i}})$$

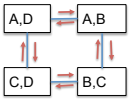
BP on a clique tree

- Messages start at leaves, propagate up to root, then propagate down to leaves



BP on loopy cluster graph

- Nodes send messages to their neighbors, based on their clique potential and messages from neighbors

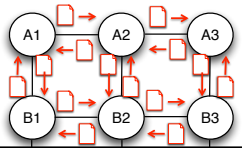


Max-product loopy BP on a grid

- Messages are passed between the variables
  - Message from node *i* to *j* at iteration *t* is,

$$m_{i\to j}^{(t)}(X_j) = \max_{X_i} \phi(X_i, X_j) \prod_{k\in\mathcal{N}(i)-\{j\}} m_{k\to i}^{(t-1)}(X_i)$$

  - Intuitively (and anthropomorphically) a message from me to you says, for each of your states, "If you choose state Xj, here's **the happiest** my neighbors and I could be."



Taking logarithms

- We can maximize logs instead of probabilities,

$$\begin{aligned}\mathbf{Y}^* &= \arg\max_{\mathbf{Y}} P(\mathbf{X},\mathbf{Y}) \\ &= \arg\max_{\mathbf{Y}} \prod \phi(\cdot) \\ &= \arg\max_{\mathbf{Y}} \log\left(\prod \phi(\cdot)\right) \\ &= \arg\max_{\mathbf{Y}} \left(\sum \log \phi(\cdot)\right)\end{aligned}$$

  - Thus all the Max-product algorithms we've seen have a corresponding Max-sum version in log space

Faster MAP inference?

- We've now seen two algorithms for MAP inference:
  - Variable elimination: Exact, but potentially very slow
  - Loopy Max-product BP: Fast, but approximate
- It turns out that in some cases, MAP problems are easier than Marginal inference problems
  - One interesting case: With binary random variables, and potential functions that satisfy (relatively weak) restrictions, exact inference on a pairwise Markov network is efficient