Inference & Applications of MRFs

16.01.2014

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de
leibe@vision.rwth-aachen.de

Announcements

- Seminar in the summer semester
  - “Current Topics in Computer Vision and Machine Learning”
  - Block seminar, presentations in 1st week of semester break
  - You can sign up for the seminar here: https://www.graphics.rwth-aachen.de/apse
- Quick poll: Who would be interested in that?

Course Outline

- Fundamentals (2 weeks)
  - Bayes Decision Theory
  - Probability Density Estimation
- Discriminative Approaches (5 weeks)
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees
- Generative Models (4 weeks)
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference
  - Applications

Topics of This Lecture

- Recap: Exact inference
  - Sum-Product algorithm
  - Max-Sum algorithm
  - Junction Tree algorithm
- Applications of Markov Random Fields
  - Application examples from computer vision
  - Interpretation of clique potentials
  - Unary potentials
  - Pairwise potentials
- Solving MRFs with Graph Cuts
  - Graph cuts for image segmentation
  - s-t mincut algorithm
  - Extension to non-binary case
  - Applications

Recap: Factor Graphs

- Joint probability
  - Can be expressed as product of factors:
    \[ p(x) = \frac{1}{Z} \prod f_s(x_s) \]
  - Factor graphs make this explicit through separate factor nodes.
- Converting a directed polytree
  - Conversion to undirected tree creates loops due to moralization!
  - Conversion to a factor graph again results in a tree!

Recap: Sum-Product Algorithm

- Objectives
  - Efficient, exact inference algorithm for finding marginals.
- Procedure:
  - Pick an arbitrary node as root.
  - Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
  - Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
  - Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.
    \[ p(x) \propto \prod_{s \in \omega(x)} \mu_{f_s \rightarrow x}(x) \]
- Computational effort
  - Total number of messages = 2 . number of graph edges.
Recap: Sum-Product Algorithm

- Two kinds of messages
  - Message from factor node to variable nodes:
    - Sum of factor contributions
    \[ \mu_{f \rightarrow x} (x) = \sum_{X_s} \prod_{m \in \text{msg}(f_i)} \mu_{x_m \rightarrow f_i} (x_m) \]
  - Message from variable node to factor node:
    - Product of incoming messages
    \[ \mu_{x_m \rightarrow f_i} (x_m) = \prod_{l \in \text{msg}(x_m) \setminus f_i} \mu_{f_l \rightarrow x_m} (x_m) \]

⇒ Simple propagation scheme.

Recap: Max-Sum Algorithm

- Objective: an efficient algorithm for finding
  - Value \( x_{\max} \) that maximizes \( p(x) \);
  - Value of \( p(x_{\max}) \).

⇒ Application of dynamic programming in graphical models.

- Key ideas
  - We are interested in the maximum value of the joint distribution
    \[ p(x_{\max}) = \max_x p(x) \]
  - For numerical reasons, use the logarithm.
    \[ \ln \left( \max_x p(x) \right) = \max_x \ln p(x) \]

⇒ Maximize the sum (of log-probabilities).

Recap: Max-Sum Algorithm

- Initialization (leaf nodes)
  \[ \mu_{f \rightarrow x} (x) = 0 \quad \mu_{f \rightarrow x} (x) = \ln f(x) \]

- Recursion
  - Messages
    \[ \mu_{f \rightarrow x} (x) = \max_{x_{\max}} \left[ \ln f(x, x_1, \ldots, x_M) + \sum_{m \in \text{msg}(f_i)} \mu_{x_m \rightarrow f_i} (x_m) \right] \]
    \[ \mu_{x_m \rightarrow f_i} (x_m) = \sum_{l \in \text{msg}(x_m) \setminus f_i} \mu_{f_l \rightarrow x_m} (x_m) \]
  - For each node, keep a record of which values of the variables gave rise to the maximum state:
    \[ \phi(x) = \arg \max_{x_{\max}} \left[ \ln f(x, x_1, \ldots, x_M) + \sum_{m \in \text{msg}(x_m) \setminus f_i} \mu_{x_m \rightarrow f_i} (x_m) \right] \]

- Termination (root node)
  - Score of maximal configuration
    \[ p_{\max} = \max \left[ \sum_{x} \mu_{f \rightarrow x} (x) \right] \]
  - Value of root node variable giving rise to that maximum
    \[ x_{\max} = \arg \max_k \left[ \sum_{x} \mu_{f \rightarrow x} (x) \right] \]
  - Back-track to get the remaining variable values
    \[ x_{\max}^{(n)} = \phi(x_{\max}) \]
Topics of This Lecture

- Factor graphs
  - Construction
  - Properties
- Sum-Product Algorithm for computing marginals
  - Key ideas
  - Derivation
  - Example
- Max-Sum Algorithm for finding most probable value
  - Key ideas
  - Derivation
  - Example
- Algorithms for loopy graphs
  - Junction Tree algorithm
  - Loopy Belief Propagation

Junction Tree Algorithm

- Motivation
  - Exact inference on general graphs.
  - Works by turning the initial graph into a junction tree and then running a sum-product-like algorithm.
  - Intractable on graphs with large cliques.
- Main steps
  1. If starting from directed graph, first convert it to an undirected graph by moralization.
  2. Introduce additional links by triangulation in order to reduce the size of cycles.
  3. Find cliques of the moralized, triangulated graph.
  4. Construct a new graph from the maximal cliques.
  5. Remove minimal links to break cycles and get a junction tree.
  \[ \Rightarrow \] Apply regular message passing to perform inference.

1. Convert to an undirected graph through moralization.
   - Marry the parents of each node.
   - Remove edge directions.

2. Triangulate
   - Such that there is no loop of length \( > 3 \) without a chord.
   - This is necessary so that the final junction tree satisfies the "running intersection" property (explained later).

3. Find cliques of the moralized, triangulated graph.
4. Construct a new junction graph from maximal cliques.
   - Create a node from each clique.
   - Each link carries a list of all variables in the intersection.
     Drawn in a "separator" box.

5. Remove links to break cycles ⇒ junction tree.
   - For each cycle, remove the link(s) with the minimal number of
     shared nodes until all cycles are broken.
   - Result is a maximal spanning tree, the junction tree.

Junction Tree - Properties

- Running intersection property
  - "If a variable appears in more than one clique, it also appears
    in all intermediate cliques in the tree".
  - This ensures that neighboring cliques have consistent probability
    distributions.
  - Local consistency → global consistency

Interpretation of the Junction Tree

- Undirected graphical model

\[
P(U) = \prod P(Clique) / \prod P(Separator)
\]

\[
P(A,B,C) = P(A,B) P(B,C) / P(B)
\]

Junction Tree: Example 1

- Algorithm
  1. Moralization
  2. Triangulation (not necessary here)
Junction Tree: Example 1

- Algorithm
  1. Moralization
  2. Triangulation (not necessary here)
  3. Find cliques
  4. Construct junction graph
  5. Break links to get junction tree

Junction Tree: Example 2

- Without triangulation step
  - The final graph will contain cycles that we cannot break without losing the running intersection property!

Junction Tree: Example 2

- When applying the triangulation
  - Only small cycles remain that are easy to break.
  - Running intersection property is maintained.

Junction Tree Algorithm

- Good news
  - The junction tree algorithm is efficient in the sense that for a given graph there does not exist a computationally cheaper approach.

- Bad news
  - This may still be too costly.
  - Effort determined by number of variables in the largest clique.
  - Grows exponentially with this number (for discrete variables).
  - Algorithm becomes impractical if the graph contains large cliques!

Loopy Belief Propagation

- Alternative algorithm for loopy graphs
  - Sum-Product on general graphs.
  - Strategy: simply ignore the problem.
  - Initial unit messages passed across all links, after which messages are passed around until convergence
    - Convergence is not guaranteed!
    - Typically break off after fixed number of iterations.
  - Approximate but tractable for large graphs.
  - Sometime works well, sometimes not at all.

Topics of This Lecture

- Recap: Exact inference
  - Sum-Product algorithm
  - Max-Sum algorithm
  - Junction Tree algorithm

- Applications of Markov Random Fields
  - Application examples from computer vision
  - Interpretation of clique potentials
  - Unary potentials
  - Pairwise potentials

- Solving MRFs with Graph Cuts
  - Graph cuts for image segmentation
  - s-t mincut algorithm
  - Extension to non-binary case
  - Applications
Markov Random Fields (MRFs)

- What we’ve learned so far...
  - We know they are undirected graphical models.
  - Their joint probability factorizes into clique potentials,
    \[ p(x) = \frac{1}{Z} \prod_i \psi_i(x_i) \]
    which are conveniently expressed as energy functions.
    \[ \psi_i(x_i) = \exp\{-E(x_i)\} \]
  - We know how to perform inference for them.
    - Sum/Max-Product BP for exact inference in tree-shaped MRFs.
    - Loopy BP for approximate inference in arbitrary MRFs.
    - Junction Tree algorithm for converting arbitrary MRFs into trees.
- But what are they actually good for?
  - And how do we apply them in practice?

Markov Random Fields

- Allow rich probabilistic models.
  - But built in a local, modular way.
  - Learn local effects, get global effects out.
- Very powerful when applied to regular structures.
  - Such as images...

Applications of MRFs

- Movie “No Way Out” (1987)
- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting

Observation process

Noisy observations

“Smoothness constraints”

“True” image content

Results by Roth & Black, CVPR’05

Observed evidence

Hidden “true states”

Neighborhood relations

Results adapted from William Freeman

Applications of MRFs
Applications of MRFs

• Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration
  - Image segmentation

Convert a low-res image into a high-res image!

Applications of MRFs

• Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration
  - Image segmentation

Super-resolution

Applications of MRFs

• Many applications for low-level vision tasks
  - Image denoising
  - Super-resolution
  - Optical flow
  - Image segmentation

Stereo depth estimation

Applications of MRFs

• Many applications for low-level vision tasks
  - Image denoising
  - Super-resolution
  - Optical flow
  - Stereo depth estimation

Stereo image pair

Disparity map
Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Super-resolution
  - Inpainting
  - Optical flow
  - Image restoration
  - Stereo depth estimation
  - Image segmentation

- MRFs have become a standard tool for such tasks.
  - Let’s look at how they are applied in detail...

MRF Structure for Images

- Basic structure

  - Two components
    - Observation model
      - How likely is it that node $x_i$ has label $L_i$ given observation $y_i$?
      - This relationship is usually learned from training data.
    - Neighborhood relations
      - Simplest case: 4-neighborhood
      - Serve as smoothing terms.
      - Discourage neighboring pixels to have different labels.
      - This can either be learned or be set to fixed “penalties”.

MRF Nodes as Pixels

- Original image
- Degraded image
- Reconstruction from MRF modeling pixel neighborhood statistics

These neighborhood statistics can be learned from training data!

MRF Nodes as Patches

- Image patches
- Scene patches

More general relationships expressed by potential functions $\Phi$ and $\Psi$.

Simple Binary Image Denoising Model

- MRF Structure

  - Example: simple energy function

    $E(x, y) = h \sum_i x_i + \beta \sum_{i,j} \delta(x_i \neq x_j) - \eta \sum_i x_i y_i$

  - Prior
  - Smoothness
  - Observation

  - Smoothness term: fixed penalty $\beta$ if neighboring labels disagree.
  - Observation term: fixed penalty $\eta$ if label and observation disagree.

Network Joint Probability

- Interpretation of the factorized joint probability

  $P(x, y) = \prod_i \Phi(x_i, y_i) \prod_{i,j} \Psi(x_i, x_j)$
**Energy Formulation**

- **Energy function**
  \[ E(x, y) = \sum_{(i, j)} \phi(x_i, y_j) + \sum_{(i, j)} \psi(x_i, x_j) \]
  - Single-node potentials \( \phi \)
    - Encode local information about the given pixel/patch.
    - How likely is a pixel/patch to belong to a certain class (e.g. foreground/background)?
  - Pairwise potentials \( \psi \)
    - Encode neighborhood information.
    - How different is a pixel/patch’s label from that of its neighbor? (e.g. based on intensity/color/texture difference, edges)

- **Single-node (unary) potentials \( \phi \)**
  - \( \phi(x_i, y_j) \)
    - \( y_j \) is the label of pixel/patch \( x_i \)
  - \( \phi(x_i, y_j) \) models the likelihood of label \( y_j \) belonging to pixel/patch \( x_i \)

- **Pairwise potentials \( \psi \)**
  - \( \psi(x_i, x_j) \)
    - \( x_i \) and \( x_j \) are adjacent pixels/patches


**How to Set the Potentials? Some Examples**

- **Unary potentials**
  - E.g. color model, modeled with a Mixture of Gaussians
  - \( \phi(x_i, y_j; \theta_{yk}) = \log \sum_k p(k|x_i)N(y_j; \bar{y}_k, \Sigma_k) \)
    - \( \phi(x_i, y_j) \) models the likelihood of label \( y_j \) belonging to pixel/patch \( x_i \)
    - Learn color distributions for each label

- **Pairwise potentials**
  - Potts Model
    - Simplest discontinuity preserving model.
    - Discontinuities between any pair of labels are penalized equally.
    - Useful when labels are unordered or number of labels is small.
  - Extension: “contrast sensitive Potts model”
    - \( \psi(x_i, x_j, g_{ij}(y); \theta_{gi}) \)
      - \( g_{ij}(y) \) is a measure of the difference between labels \( y_i \) and \( y_j \)
      - Encourages smoothness where there is also a large change in the observations.


**Example: MRF for Image Segmentation**

- **MRF structure**
  - \( \phi(D|x_i, x_j) \)
    - Pairwise potential
  - \( \phi(D|x_i) \)
    - Unary potential
  - \( \phi(D) \)
    - Uniform prior

- **Energy formulation**
  - \( E(x) = \sum_{i \in D} \phi(D|x_i) + \sum_{ij \in E} (\phi(D|x_i, x_j) + \psi(x_i, x_j)) \)
  - \( \psi(x_i, x_j) \) models the likelihood of labels \( x_i \) and \( x_j \) being different


**Energy Minimization**

- **Goal:**
  - Infer the optimal labeling of the MRF.
- **Many inference algorithms are available, e.g.**
  - Simulated annealing
  - Iterated conditional modes (ICM)
  - Belief propagation
  - Graph cuts
  - Variational methods
  - Monte Carlo sampling

- **Recently, Graph Cuts have become a popular tool**
  - Only suitable for a certain class of energy functions.
  - But the solution can be obtained very fast for typical vision problems (~1MPixel/sec).
References and Further Reading

- A gentle introduction to Graph Cuts can be found in the following paper:

- Try the GraphCut implementation at http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software.html