Computer Vision

Lecture 5 – Probabilistic Graphical Models

Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group

University of Tübingen / MPI-IS

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

MAX-PLANCK-GESELLSCHAFT

ellis

European Laboratory for Learning and Intelligent Systems

Agenda

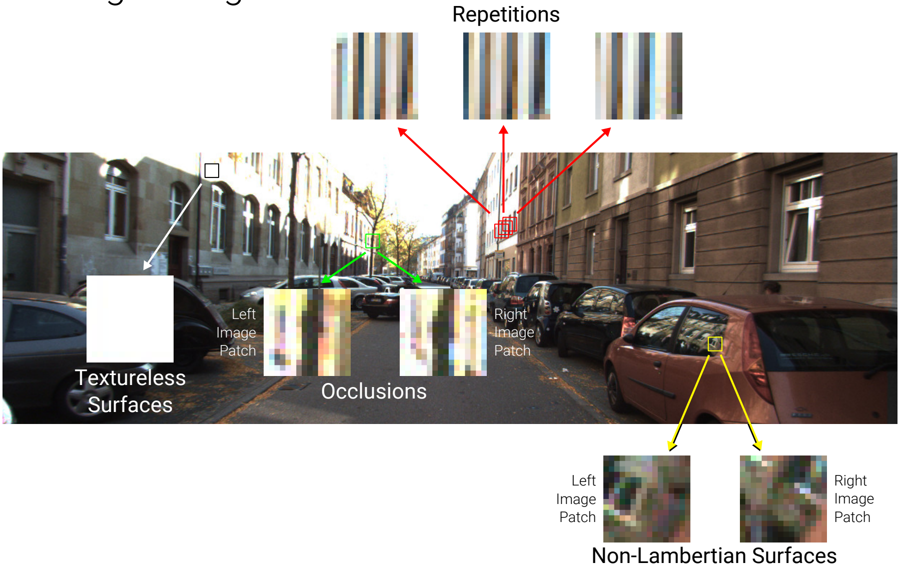
5.1
Structured Prediction

Block Matching Ambiguities

Block Matching Assumption:

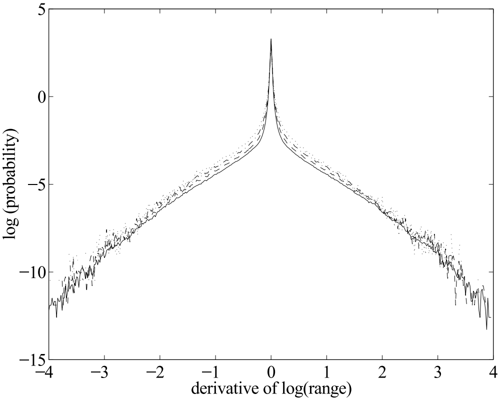
- Corresponding regions in both images look similar
- When will this similarity constraint fail?

Block Matching Ambiguities



How does the real world look like?

- Analyze real-world statistics, e.g., Brown range image database
- Conclusion: Depth varies slowly except at object discontinuities which are sparse

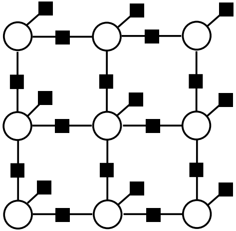


Huang, Lee and Mumford: Statistics of Range Images. CVPR, 2000.

Spatial Regularization

- How can we integrate this knowledge about the statistics of depth maps?
- Formulate problem as inference in a graphical model where each node (=variable) corresponds to one pixel and model interactions between adjacent pixels

$$p(\mathbf{D}) \propto \exp\left\{-\sum_i \psi_{data}(d_i) - \lambda \sum_{i\sim j} \psi_{smooth}(d_i, d_j)\right\}$$



- $i \sim j$: neighboring pixels (on a 4-connected grid)
- Unary terms: Matching cost $\psi_{data}(d)$
- Pairwise terms: Smoothness assumptions $\psi_{smooth}(d, d')$

Probabilistic Graphical Models

Probabilistic Graphical Models:

- Take **probabilistic view** and **model dependency structure** of the problem
- **Structured prediction** based on **local constraints** between random variables
- Graphical models have ruled computer vision before the deep learning revolution
- Useful in the presence of **little training data** to **integrate prior knowledge**
- Graphical models can be combined with / can inform deep learning (Lecture 7)

Pros:

- Integration of prior knowledge
- Few parameters, limited data
- Interpretable models by design

Cons:

- Many phenomena hard to model
- Exploiting large datasets difficult
- Inference often approximate

Structured Prediction

Classification / Regression:

$$f : \mathbb{X} \to \mathbb{N} \qquad or \qquad f : \mathbb{X} \to \mathbb{R}$$

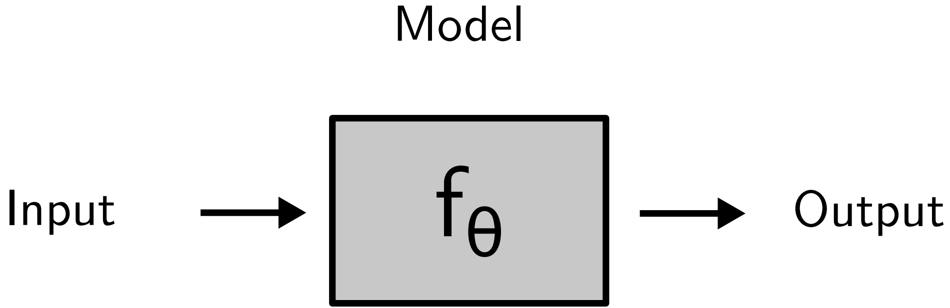
- Inputs $\mathcal{X} \in \mathbb{X}$ can be any kind of objects
  - images, text, audio, sequence of amino acids, ...
- Output $y \in \mathbb{N} / y \in \mathbb{R}$ is a discrete or real number
  - classification, regression, density estimation, ...

Structured Prediction:

$$f : \mathbb{X} \to \mathbb{Y}$$

- Inputs $\mathcal{X} \in \mathbb{X}$ can be any kind of objects
- Outputs $\mathcal{Y} \in \mathbb{Y}$ are complex (structured) objects
  - images, text, parse trees, folds of a protein, computer programs, ...

Supervised Learning



- **Learning:** Estimate parameters θ from training dataset $\{(x_i, y_i)\}_{i=1}^N$
- **Inference:** Make novel predictions $\hat{y} = f_\theta(x)$ for unseen inputs x

Classification / Regression

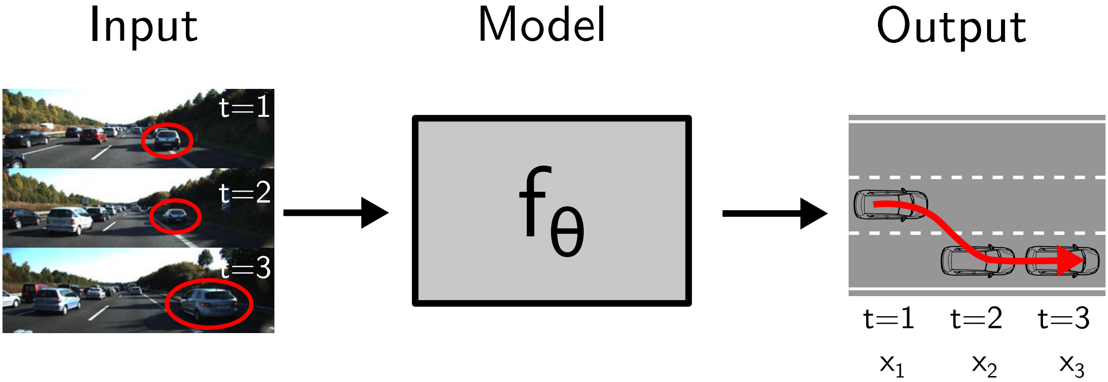
Input | Model | Output

→ f_θ → "Beach"

Classification / Regression:

- Output is a one-dimensional (discrete or continuous) variable
- Example siamese network: predict disparity independently for each pixel

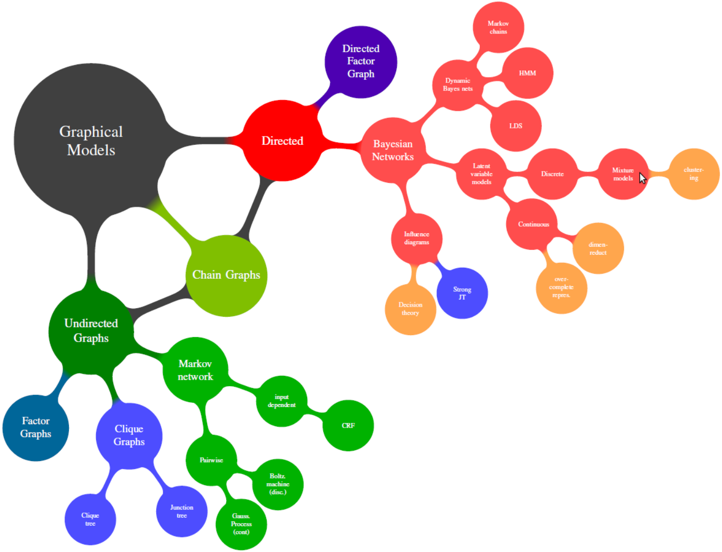
Structured Prediction



Structured Prediction:

- Probabilistic graphical models encode local dependencies of the problem
- Deep neural netwoks with image-based outputs (stereo, flow, semantics)

Probabilistic Graphical Models



Overview

Lecture 5: Probabilistic Graphical Models

- Markov Networks, Factor Graphs, Belief Propagation

Lecture 6: Applications of Graphical Models

- Stereo, Optical Flow & Multi-view Reconstruction

Lecture 7: Learning in Graphical Models

- Parameter Estimation and Deep Structured Models

Further Reading

- `http://www.nowozin.net/sebastian/cvpr2012tutorial/`
- `http://www.cs.ucl.ac.uk/staff/d.barber/brml/`

5.2
Markov Random Fields

Probability Theory Recap

Random Variables:

- Discrete random variable: $x \in \{1, \dots, C\}$
  - Probability that x takes value c: $p(x = c)$
- Continuous random variable: $x \in \mathbb{R}$
  - Probability that x takes value in $\mathbb{A} \subset \mathbb{R}$: $p(x \in \mathbb{A})$
- Distribution over x: $p(x)$ (we use lowercase p also for discrete distributions)

Properties:

- Joint distribution: $p(x, y)$ as short notation for $p(x = c, y = c')$
- Sum rule (marginal distribution): $p(x) = \sum_y p(x, y)$ or $p(x) = \int_y p(x, y)$
- Product rule: $p(x, y) = p(y|x)p(x)$
- Bayes rule: $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$

Markov Random Field

Potential

A **potential** $\phi(x)$ is a non-negative function of the variable x. A **joint potential** $\phi(x_1, x_2, \dots)$ is a non-negative function of a **set** of variables.

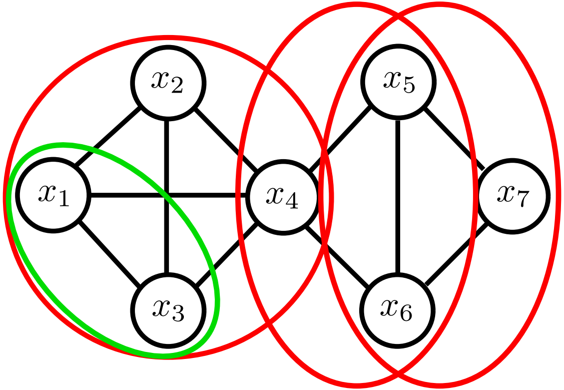
Markov Random Field (MRF) = Markov Network

For a set of variables $\mathcal{X} = \{x_1, \dots, x_M\}$ a **Markov Random Field** is defined as a product of potentials over the **(maximal) cliques** $\{\mathcal{X}_k\}_{k=1}^K$ of the undirected graph $\mathcal{G}$:

$$p(\mathcal{X}) = \frac{1}{Z} \prod_{k=1}^{K} \phi_k(\mathcal{X}_k)$$

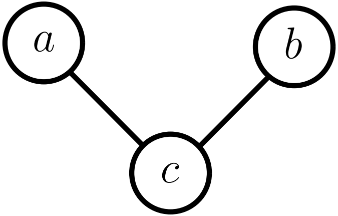
- The factorization is **not unique:** an MRF can have multiple different factorizations
- If all potentials are strictly positive: **Gibbs distribution**

Undirected Graph



- An undirected graph $\mathcal{G}$ is a graph with vertices and undirected edges
- A clique (green, red) is a subset of vertices that are fully connected
- A maximal clique (red) is a clique that cannot be extended by any other vertex

Properties of Markov Random Fields

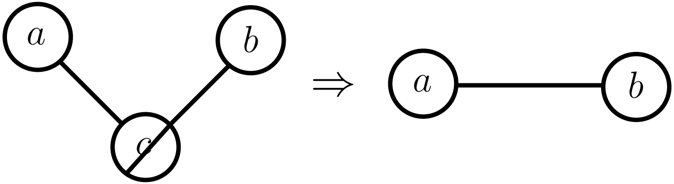


$$p(a, b, c) = \frac{1}{Z}\phi_1(a, c)\phi_2(b, c)$$

- Two maximal cliques of size two: $\mathcal{X}_1 = \{a, c\}$ and $\mathcal{X}_2 = \{b, c\}$
- Z normalizes the distribution and is called **partition function**

$$Z = \sum_{a,b,c} \phi_1(a, c)\phi_2(b, c)$$

Properties of Markov Random Fields



- Marginalizing over c makes a and b dependent
- Proof by showing that a and b are not independent:

$$p(a, b) \neq p(a)p(b)$$

Properties of Markov Random Fields

Let's show this statement by contradiction. Assume the following holds true:

$$p(a,b) = \sum_c p(a,b,c) = \frac{1}{Z}\sum_c \phi_1(a,c)\phi_2(b,c)$$

$$= p(a)p(b) = \sum_{b,c} p(a,b,c) \sum_{a,c} p(a,b,c) = \frac{1}{Z}\sum_{b,c} \phi_1(a,c)\phi_2(b,c)\ \frac{1}{Z}\sum_{a,c} \phi_1(a,c)\phi_2(b,c)$$

Therefore, we have:

$$\sum_{a,b,c} \phi_1(a,c)\phi_2(b,c) \sum_c \phi_1(a,c)\phi_2(b,c) = \sum_{b,c} \phi_1(a,c)\phi_2(b,c) \sum_{a,c} \phi_1(a,c)\phi_2(b,c)$$

Properties of Markov Random Fields

Consider binary variables $a, b, c \in \{0, 1\}$ and the following choice of potentials

$$\phi_1(a,c) = [a = c] \quad \text{and} \quad \phi_2(b,c) = [b = c]$$

where $[\cdot]$ is the Iverson bracket which takes 1 if its argument is true and 0 otherwise.

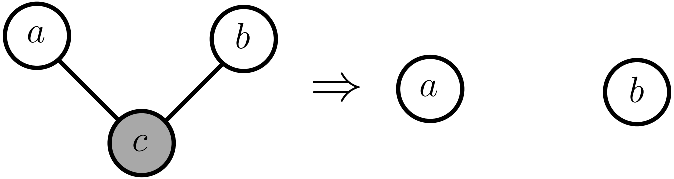
We obtain the following contradiction:

$$\underbrace{\sum_{a,b,c} \phi_1(a,c)\phi_2(b,c)}_{2} \underbrace{\sum_{c} \phi_1(a,c)\phi_2(b,c)}_{[a=b]} = \underbrace{\sum_{b,c} \phi_1(a,c)\phi_2(b,c)}_{1} \underbrace{\sum_{a,c} \phi_1(a,c)\phi_2(b,c)}_{1}$$

Therefore, in general (for arbitrary choices of the potentials):

$$p(a,b) \neq p(a)p(b)$$

Properties of Markov Random Fields



- Conditioning on c makes a and b independent
- We write this conditional independence statement compactly as: $a \perp\!\!\!\perp b \mid c$
- Proof by showing (exercise):

$$p(a, b \mid c) = p(a \mid c)p(b \mid c)$$

Global Markov Property

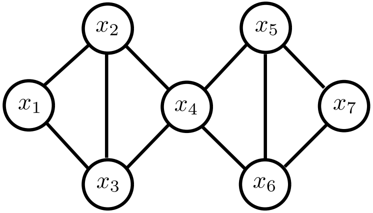
The examples above can be generalized, yielding the **global Markov property:**

Separation

A subset $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$ if every path from a member of $\mathcal{A}$ to any member of $\mathcal{B}$ passes through $\mathcal{S}$.

Global Markov Property

For disjoint sets of variables $(\mathcal{A}, \mathcal{B}, \mathcal{S})$ where $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$, we have $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{S}$



Local Markov Property

From the global Markov property, we can derive the **local Markov property:**

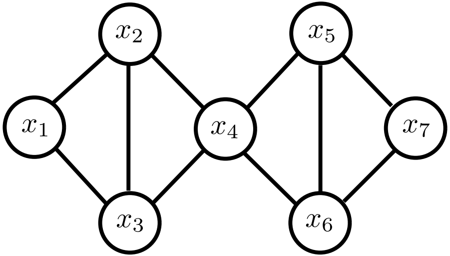
Local Markov Property

When conditioned on its neighbors, x becomes independent of the remaining variables of the graph:

$$p(x \mid \mathcal{X} \setminus \{x\}) = p(x \mid ne(x))$$

- The set of neighboring nodes $ne(x)$ is called **Markov blanket**
- This also holds for sets of variables

Local Markov Property – Example



- $p(x_4 \mid x_1, x_2, x_3, x_5, x_6, x_7) = p(x_4 \mid x_2, x_3, x_5, x_6)$
- In other words $x_4 \perp\!\!\!\perp \{x_1, x_7\} \mid \{x_2, x_3, x_5, x_6\}$
- Similarly, other independence relationships can be read off the graph

Hammersley-Clifford Theorem

Hammersley-Clifford Theorem

A probability distribution that has a **strictly positive mass** or density satisfies the **Markov properties** with respect to an undirected graph $\mathcal{G}$ if and only if it is a Gibbs random field, i.e., its density can be **factorized** over the (maximal) cliques of the graph.

Filter View of Graphical Models:

- Only distributions that factorize based on the (maximal) cliques may pass
- Alternatively, only distributions that respect the Markov properties may pass
- From the theorem above we know that both sets of distributions are identical

Clifford: Markov random fields in statistics. Oxford University Press, 1990.

5.3

Factor Graphs

MRF Factorization Ambiguities

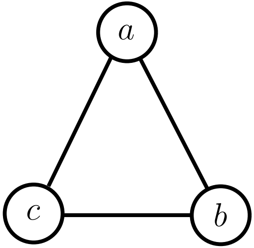
- Consider this factorization into potential functions:

$$p(a,b,c) = \frac{1}{Z}\phi(a,b)\phi(b,c)\phi(c,a)$$

- Which other factorization is represented by this Markov network?

$$p(a,b,c) = \frac{1}{Z}\phi(a,b,c)$$

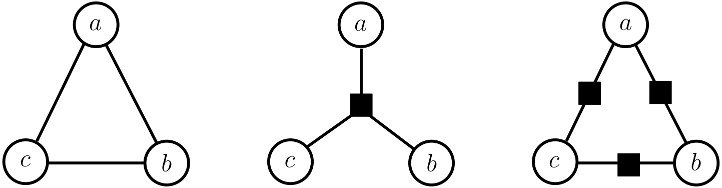
- What is the corresponding Markov network / MRF?



- The second factorization is more general (admits larger class of distributions)
- But both factorizations respect the **same cond. independence assumptions**
- Thus, the **factorization** is **not uniquely specified** by the Markov network / graph

Factor Graphs

To disambiguate, we introduce an extra node (a square) for each factor:



- Left: Markov Network of $\frac{1}{Z}\phi(a,b,c)$ and $\frac{1}{Z}\phi(a,b)\phi(b,c)\phi(c,a)$
- Middle: Factor graph representation of $\frac{1}{Z}\phi(a,b,c)$
- Right: Factor graph representation of $\frac{1}{Z}\phi(a,b)\phi(b,c)\phi(c,a)$
- The two factor graphs correspond to the **same Markov network**
- But they allow to distinguish different **factorizations** by making them **explicit**

Factor Graphs

Factor Graph

Given $\mathcal{X} = \{x_1, \ldots, x_M\}$, $\{\mathcal{X}_k\}_{k=1}^K$ with $\mathcal{X}_k \subseteq \mathcal{X}$ and a function

$$f(\mathcal{X}) = \prod_{k=1}^{K} f_k(\mathcal{X}_k)$$

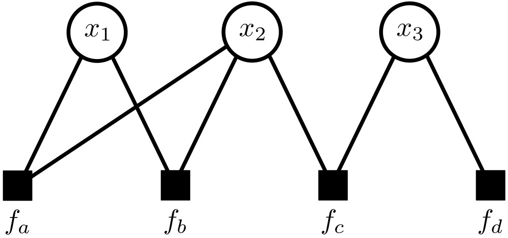
the **factor graph (FG)** is a bipartite graph with a **square node** for each factor f_k and a **circle node** for each variable x_i. By normalizing $f(\cdot)$, we obtain a distribution:

$$p(\mathcal{X}) = \frac{1}{Z} \prod_{k=1}^{K} f_k(\mathcal{X}_k)$$

- As in the previous unit, $Z = \sum_{\mathcal{X}} f(\mathcal{X})$ denotes the partition function

Factor Graph: Example 1

- Question: which distribution ?



- Answer:

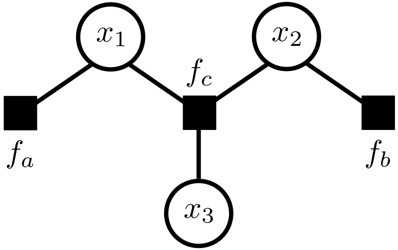
$$p(x_1, x_2, x_3) = \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

Factor Graph: Example 2

- Question: Which factor graph?

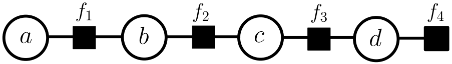
$$p(x_1, x_2, x_3) = p(x_1)\, p(x_2)\, p(x_3|x_1, x_2)$$

- Answer:



5.4
Belief Propagation

Inference in Chain Structured Factor Graphs

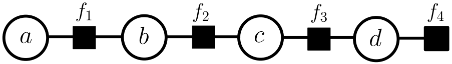


$$p(a,b,c,d) = \frac{1}{Z} f_1(a,b) f_2(b,c) f_3(c,d) f_4(d)$$

$$p(a) = \sum_{b,c,d} p(a,b,c,d) = ?$$

Computational Complexity: $C^{M-1} = 2^3 = 8$ (if variables are binary)

Inference in Chain Structured Factor Graphs

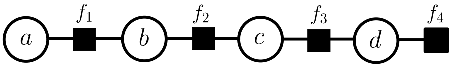


$$p(a,b,c,d) = \frac{1}{Z} f_1(a,b) f_2(b,c) f_3(c,d) f_4(d)$$

$$\begin{aligned} p(a,b,c) &= \sum_d p(a,b,c,d) \\ &= \frac{1}{Z} f_1(a,b) f_2(b,c) \underbrace{\sum_d f_3(c,d) f_4(d)}_{\mu_{d \to c}(c)} \end{aligned}$$

$$p(a,b) = \sum_c p(a,b,c) = \frac{1}{Z} f_1(a,b) \underbrace{\sum_c f_2(b,c) \mu_{d \to c}(c)}_{\mu_{c \to b}(b)}$$

Inference in Chain Structured Factor Graphs



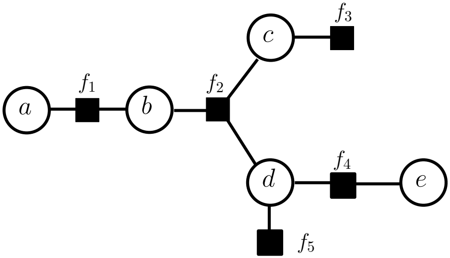
- Simply recurse further:

$$p(a) = \sum_b p(a,b) = \frac{1}{Z} \sum_b f_1(a,b)\, \mu_{c\to b}(b) = \frac{1}{Z} \mu_{b\to a}(a)$$

- $\mu_{m\to n}(n)$ carries the information beyond m
- Computational complexity? $(M-1)\cdot C = 3\cdot 2 = 6$ (if variables are binary)
- We did not need the factors yet
- But we will see that making a distinction is helpful

Inference in Tree Structured Factor Graphs

- Consider a branching graph (tree):



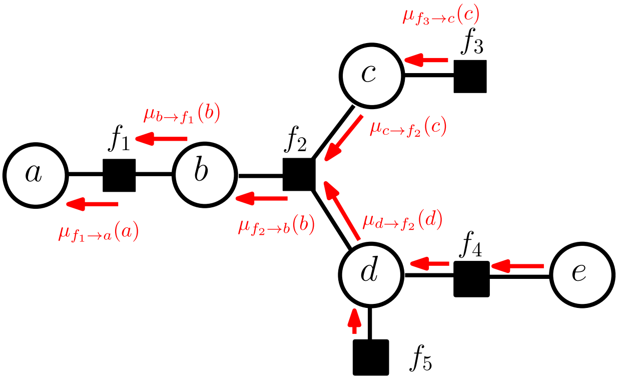
with factors

$$f_1(a,b)f_2(b,c,d)f_3(c)f_4(d,e)f_5(d)$$

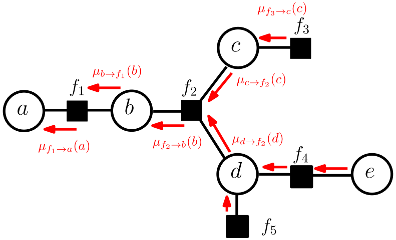
- How to compute the marginal distribution $p(a,b)$?

Inference in Tree Structured Factor Graphs

- Idea: Compute and pass messages



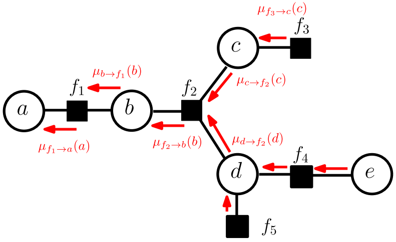
Inference in Tree Structured Factor Graphs



$$p(a,b) = \frac{1}{Z} f_1(a,b) \underbrace{\sum_{c,d,e} f_2(b,c,d) f_3(c) f_5(d) f_4(d,e)}_{\mu_{f_2 \to b}(b)}$$

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b,c,d) f_3(c) f_5(d) \sum_{e} f_4(d,e)$$

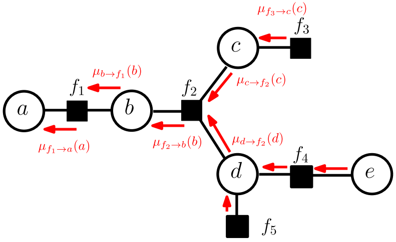
Inference in Tree Structured Factor Graphs



$$p(a,b) = \frac{1}{Z} f_1(a,b) \underbrace{\sum_{c,d,e} f_2(b,c,d) f_3(c) f_5(d) f_4(d,e)}_{\mu_{f_2 \to b}(b)}$$

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b,c,d) \underbrace{f_3(c)}_{\mu_{c \to f_2}(c)} \underbrace{f_5(d) \sum_{e} f_4(d,e)}_{\mu_{d \to f_2}(d)}$$

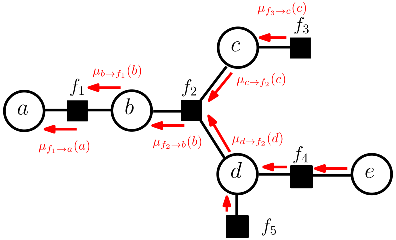
Inference in Tree Structured Factor Graphs



$$p(a,b) = \frac{1}{Z} f_1(a,b) \underbrace{\sum_{c,d,e} f_2(b,c,d) f_3(c) f_5(d) f_4(d,e)}_{\mu_{f_2 \to b}(b)}$$

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b,c,d) \mu_{c \to f_2}(c) \mu_{d \to f_2}(d)$$

Factor-to-Variable Messages



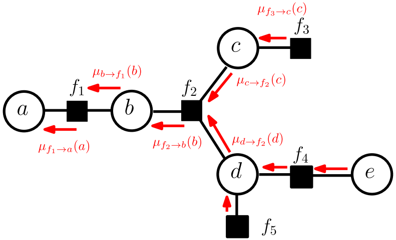
- Repeated from last slide:

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b, c, d) \mu_{c \to f_2}(c) \mu_{d \to f_2}(d)$$

- More general: for leaf factors $\mu_{f \to x}(x) = f(x)$ and otherwise:

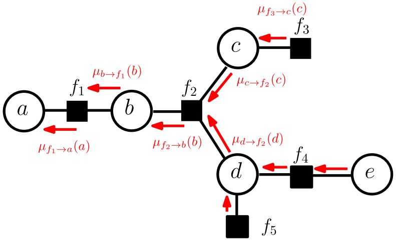
$$\mu_{f \to x}(x) = \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y \to f}(y)$$

Variable-to-Factor Messages



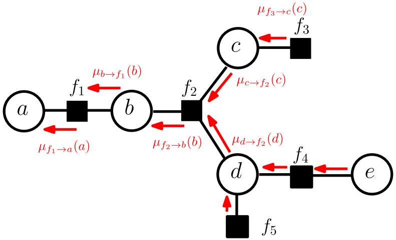
$$\mu_{d \to f_2}(d) = f_5(d) \sum_e f_4(d, e)$$

Variable-to-Factor Messages



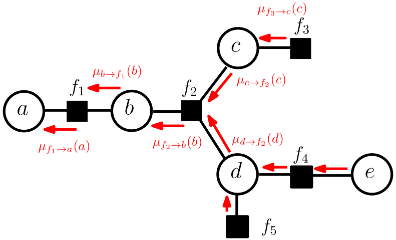
$$\mu_{d\to f_2}(d) = \underbrace{f_5(d)}_{\mu_{f_5\to d}(d)} \underbrace{\sum_e f_4(d,e)}_{\mu_{f_4\to d}(d)}$$

Variable-to-Factor Messages



$$\mu_{d\to f_2}(d) = \mu_{f_5\to d}(d)\mu_{f_4\to d}(d)$$

Variable-to-Factor Messages



- Here (repeated from last slide):

$$\mu_{d\to f_2}(d) = \mu_{f_5\to d}(d)\mu_{f_4\to d}(d)$$

- General:

$$\mu_{x\to f}(x) = \prod_{g\in\{ne(x)\backslash f\}} \mu_{g\to x}(x)$$

Comments

- Many subscripts, don't get confused :)
- Once computed, messages can be re-used
- Important observation: All marginals $(p(c), p(d), p(c, d), ...)$ can be written as a function of messages
- We need an algorithm to compute all messages
- For marginal inference: Sum-product algorithm

Sum-Product Algorithm

Sum-Product Algorithm – Overview

Belief Propagation:

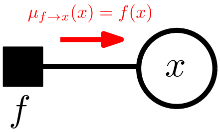
- Algorithm to compute all messages efficiently
- Assumes that the graph is singly-connected (chain, tree)

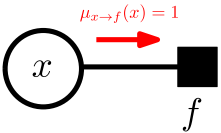
Algorithm:

1. Initialization
2. Variable to Factor message
3. Factor to Variable message
4. Repeat until all messages have been calculated
5. Calculate the desired marginals from the messages

1. Initialization

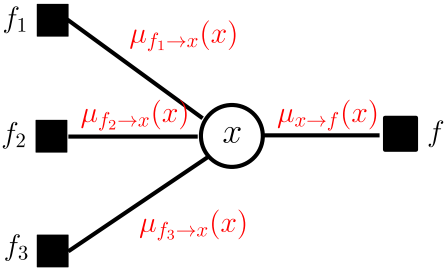
- Messages from extremal node factors are initialized to factor
- Messages from extremal variable nodes is set to 1





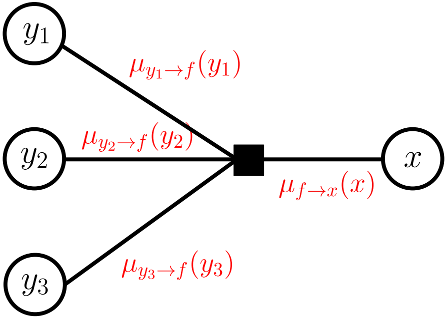
2. Variable-to-Factor Message

$$\mu_{x\to f}(x) = \prod_{g\in\{ne(x)\backslash f\}} \mu_{g\to x}(x)$$



3. Factor-to-Variable Message (Sum-Product)

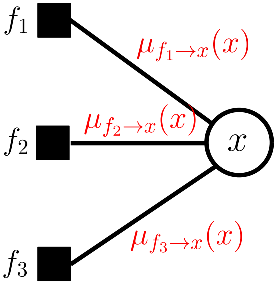
$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f \backslash x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \backslash x\}} \mu_{y\to f}(y)$$



- Remark: We sum over all states in the set of variables

5. Calculate Marginals

$$p(x) \propto \prod_{f \in ne(x)} \mu_{f \to x}(x)$$



- Remark: Unlike in step 2, here we calculate the product over all neighbors

Log Representation

- In large graphs, messages may become very small/big (due to product)
- This leads to numerical problems when storing them as floating point numbers
- Solution: work with log-messages instead $\lambda = \log \mu$
- Variable-to-factor messages

$$\mu_{x \to f}(x) = \prod_{g \in \{ne(x) \setminus f\}} \mu_{g \to x}(x)$$

then becomes

$$\lambda_{x \to f}(x) = \sum_{g \in \{ne(x) \setminus f\}} \lambda_{g \to x}(x)$$

Log Representation

- Work with log-messages instead $\lambda = \log \mu$
- Factor-to-variable messages

$$\mu_{f \to x}(x) = \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y \to f}(y)$$

then become

$$\lambda_{f \to x}(x) = \log \left(\sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \exp \left[\sum_{y \in \{ne(f) \setminus x\}} \lambda_{y \to f}(y) \right] \right)$$

Max-Product Algorithm

Finding the maximal state: Max-Product

- For a given distribution $p(a,b,c,d)$ find the most likely state:

$$a^*, b^*, c^*, d^* = \underset{a,b,c,d}{\operatorname{argmax}} \; p(a,b,c,d)$$

- This is called the **Maximum-A-Posteriori (MAP)** solution
- Again use factorization structure to distribute maximisation to local computations
- Example: Chain

$$p(a,b,c,d) = \frac{1}{Z} f_1(a,b) f_2(b,c) f_3(c,d)$$

Example: Chain

$$
\begin{aligned}
\max_{a,b,c,d} p(a,b,c,d) &= \max_{a,b,c,d} f_1(a,b) f_2(b,c) f_3(c,d) \\
&= \max_{a,b,c} f_1(a,b) f_2(b,c) \underbrace{\max_d f_3(c,d)}_{\mu_{d\to c}(c)} \\
&= \max_{a,b} f_1(a,b) \underbrace{\max_c f_2(b,c) \mu_{d\to c}(c)}_{\mu_{c\to b}(b)} \\
&= \max_a \underbrace{\max_b f_1(a,b) \mu_{c\to b}(b)}_{\mu_{b\to a}(a)} \\
&= \max_a \mu_{b\to a}(a)
\end{aligned}
$$

- We obtain the maximum probability value, but not the most probable state

Example: Chain

- Solution: Once messages are computed, find the optimal values:

$$
\begin{aligned}
a^* &= \operatorname*{argmax}_a \; \mu_{b\to a}(a) \\
b^* &= \operatorname*{argmax}_b \; f_1(a^*, b)\, \mu_{c\to b}(b) \\
c^* &= \operatorname*{argmax}_c \; f_2(b^*, c)\, \mu_{d\to c}(c) \\
d^* &= \operatorname*{argmax}_d \; f_3(c^*, d)
\end{aligned}
$$

- This is called **backtracking** (dynamic programming)
- If maximum unique the MAP solution is the maximum of the “max-marginals”
- The latter is easy to compute (find maximum of computed vector per variable)

Max-Product Algorithm – Overview

Belief Propagation:

- Algorithm to compute all messages efficiently
- Assumes that the graph is singly-connected (chain, tree)

Algorithm:

1. Initialization
2. Variable to Factor message
3. Factor to Variable message
4. Repeat until all messages have been calculated
5. Calculate the desired MAP solution

Loopy Belief Propagation

Loopy Belief Propagation

$$\mu_{x \to f}(x) = \prod_{g \in \{ne(x) \setminus f\}} \mu_{g \to x}(x)$$

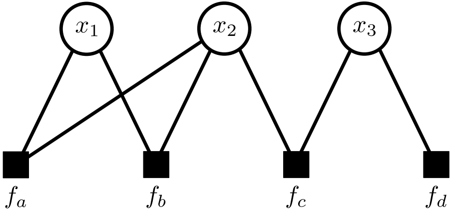
$$\mu_{f \to x}(x) = \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y \to f}(y)$$

- Messages are also well defined for loopy graphs!
- Simply apply them to loopy graphs as well
- We loose exactness ($\Rightarrow$ approximate inference)
- Even no guarantee of convergence [Yedida et al. 2004]
- But often works surprisingly well in practice

Loopy Belief Propagation

Which message passing schedule?

- Random or fixed order
- Popular choice:
  1. Factors $\rightarrow$ variables
  2. Variables $\rightarrow$ factors
  3. Repeat for N iterations
- Can be run in parallel as factor graph is bipartite:



Summary

Sum-Product Belief Propagation

Goal: Compute marginals of distribution

- Factor-to-variable messages:

$$\lambda_{f\to x}(x) = \log\left(\sum_{\mathcal{X}_f\setminus x} f(\mathcal{X}_f)\exp\left\{\sum_{y\in\{ne(f)\setminus x\}} \lambda_{y\to f}(y)\right\}\right) \qquad (1)$$

- Variable-to-factor messages:

$$\lambda_{x\to f}(x) = \sum_{g\in\{ne(x)\setminus f\}} \lambda_{g\to x}(x) \qquad (2)$$

- $\sum_{\mathcal{X}_f\setminus x}$: Summation over all states of $\mathcal{X}_f \setminus x$ (Eq. 1)
- $\sum_{y\in\{ne(f)\setminus x\}}$ / $\sum_{g\in\{ne(x)\setminus f\}}$: Summation over all incoming messages
- To avoid large values, subtract mean from $\lambda_{x\to f}(x)$ after message update (Eq. 2)

Max-Product Belief Propagation

Goal: Find most likely state (MAP state)

- Factor-to-variable messages:

$$\lambda_{f\rightarrow x}(x) = \max_{\mathcal{X}_f \backslash x}\left[\log f(\mathcal{X}_f) + \sum_{y\in\{ne(f)\backslash x\}} \lambda_{y\rightarrow f}(y)\right] \qquad (3)$$

- Variable-to-factor messages:

$$\lambda_{x\rightarrow f}(x) = \sum_{g\in\{ne(x)\backslash f\}} \lambda_{g\rightarrow x}(x) \qquad (2)$$

- $\max_{\mathcal{X}_f \backslash x}$: Maximization over all states of $\mathcal{X}_f \setminus x$ (Eq. 3)
- $\sum_{y\in\{ne(f)\backslash x\}}$ / $\sum_{g\in\{ne(x)\backslash f\}}$: Summation over all incoming messages
- To avoid large values, subtract mean from $\lambda_{x\rightarrow f}(x)$ after message update (Eq. 2)

Special Case: Pairwise MRF

Factor-to-variable messages simplify as follows.

- **Sum-Product Belief Propagation:**
  - Unary factor $f(x)$:

$$\lambda_{f\to x}(x) = \log f(x) \qquad (1)$$

  - Pairwise factor $f(x, y)$:

$$\lambda_{f\to x}(x) = \log\left(\sum_{y} f(x, y)\exp\left\{\lambda_{y\to f}(y)\right\}\right) \qquad (1)$$

Special Case: Pairwise MRF

Factor-to-variable messages simplify as follows.

- **Max-Product Belief Propagation:**
  - Unary factor $f(x)$:

$$\lambda_{f \to x}(x) = \log f(x) \qquad (3)$$

  - Pairwise factor $f(x, y)$:

$$\lambda_{f \to x}(x) = \max_{y} \left[\log f(x, y) + \lambda_{y \to f}(y)\right] \qquad (3)$$

Readout

Read off marginal or MAP state at each variable:

- Similar to variable-to-factor messages
- However: summing over **all** incoming messages

$$p(x) = \exp\{\lambda(x)\} / \sum_x \exp\{\lambda(x)\} \quad (4)$$

$$x^* = \operatorname*{argmax}_x \sum_{g \in \{ne(x)\}} \lambda_{g \to x}(x) \quad (5)$$

with $\lambda(x) = \sum_{g \in \{ne(x)\}} \lambda_{g \to x}(x)$

Algorithm Overview

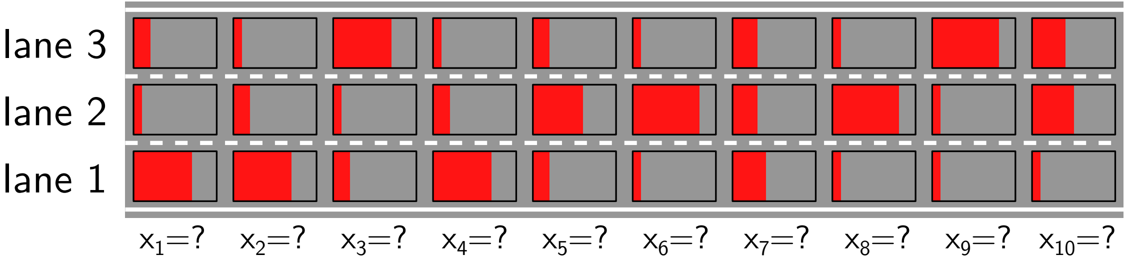
Belief Propagation Algorithm

- Input: `variables` and `factors`
- Allocate all `messages` (log representation)
- Initialize `messages` to 0 (=uniform distribution)
- For N iterations do
  - Update all `factor-to-variable messages` (Eq. 1 or Eq. 3)
  - Update all `variable-to-factor messages` (Eq. 2)
  - Normalize all `variable-to-factor messages`:
    $\lambda_{x \rightarrow f}(x) = \lambda_{x \rightarrow f}(x) - \text{mean}\left(\lambda_{x \rightarrow f}(x)\right)$
- Read off marginal or MAP state at each variable (Eq. 4 or Eq. 5)

5.5
Examples

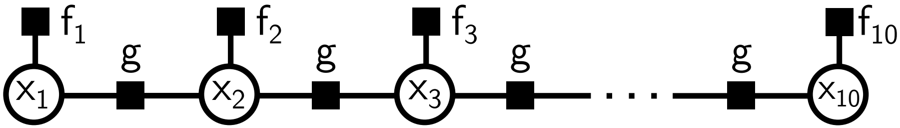
Example 1: Vehicle Localization

Example 1: Vehicle Localization



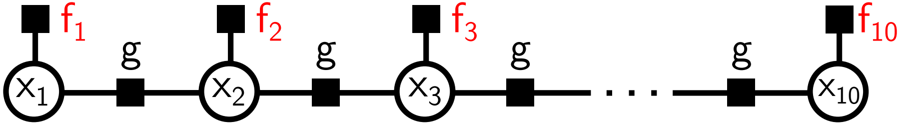
- **Goal:** Estimate vehicle location at time $t = 1, \ldots, 10$
- **Variables:** $\mathcal{X} = \{x_1, \ldots, x_{10}\} \quad x_i \in \{1, 2, 3\} \quad (C = 3)$
- **Observations:** $\mathcal{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_{10}\} \quad \mathbf{o}_i \in \mathbb{R}^3$

Example 1: Vehicle Localization



$$p_\theta(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{10} f_i(x_i) \prod_{i=1}^{9} g_\theta(x_i, x_{i+1})$$

Example 1: Vehicle Localization

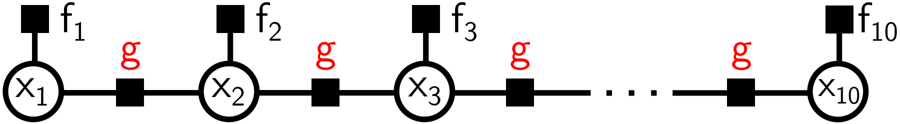


$$p_\theta(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{10} f_i(x_i) \prod_{i=1}^{9} g_\theta(x_i, x_{i+1})$$

Unary Factors:

- $f_1(x_1) = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.2 \end{bmatrix}, \quad f_2(x_2) = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}, \quad f_3(x_3) = \begin{bmatrix} 0.2 \\ 0.1 \\ 0.7 \end{bmatrix}, \quad \ldots$

Example 1: Vehicle Localization



$$p_\theta(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{10} f_i(x_i) \prod_{i=1}^{9} g_\theta(x_i, x_{i+1})$$

Pairwise Factors:

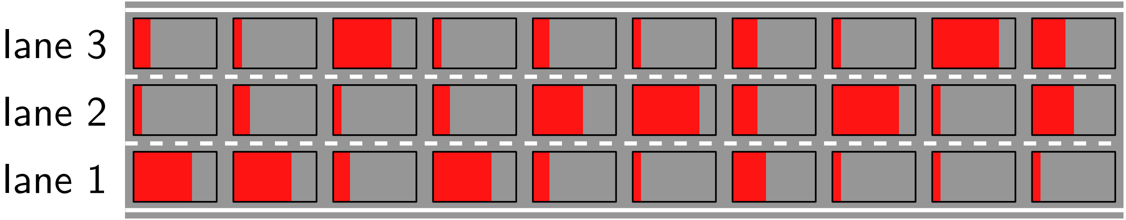
- $g_\theta(x_i, x_{i+1}) = \begin{bmatrix} 0.8 & 0.2 & 0.0 \\ 0.2 & 0.6 & 0.2 \\ 0.0 & 0.2 & 0.8 \end{bmatrix}$

- Learning Problem:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \prod_n p_\theta(\mathbf{x}_n | \mathbf{o}_n)$$

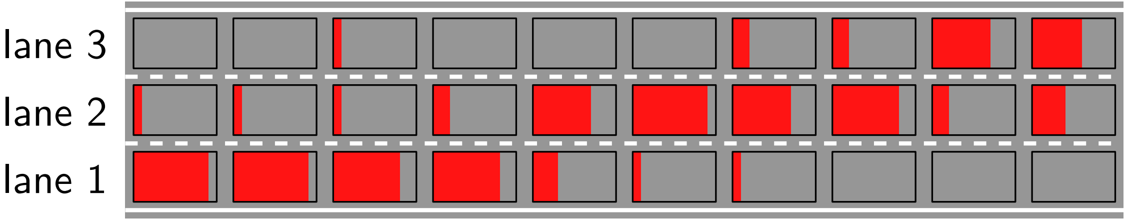
Example 1: Vehicle Localization

Observations



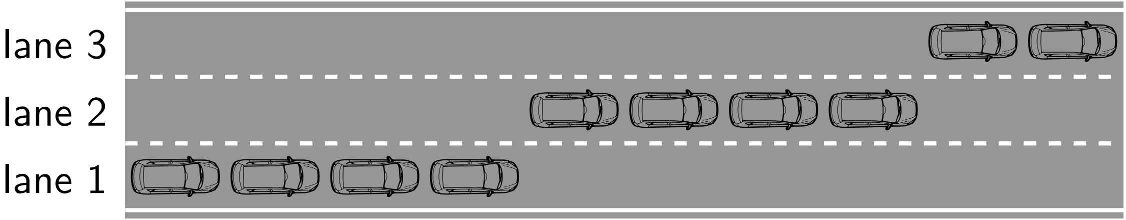
Example 1: Vehicle Localization

Marginal Distributions



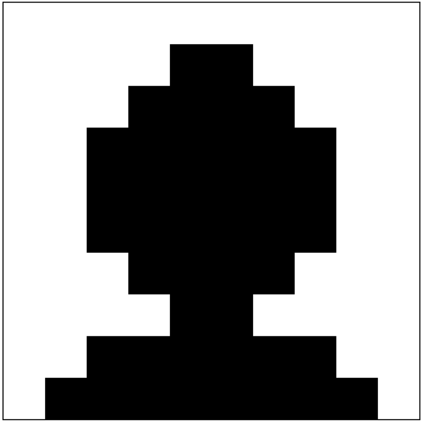
Example 1: Vehicle Localization

Maximum-A-Posteriori State

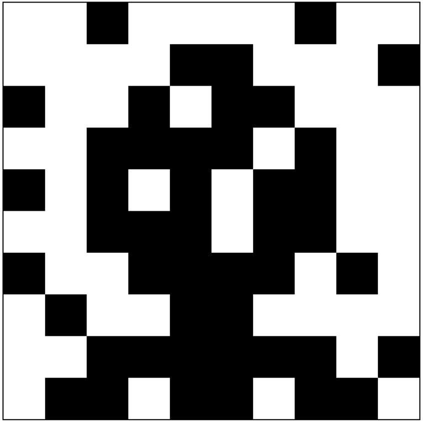


Example 2: Image Denoising

Example 2: Image Denoising

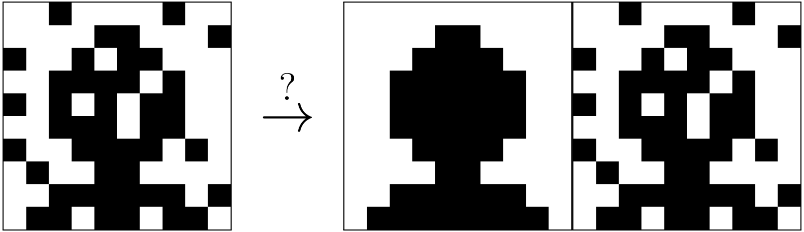


Example 2: Image Denoising



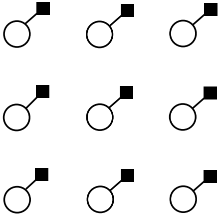
Example 2: Image Denoising

Can we recover the original image from a noisy observation?

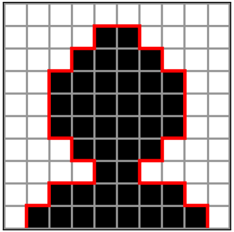


- Variables: $x_1, \ldots, x_{100} \in \{0, 1\}$
- Unary log factors: $\psi_1(x_1), \ldots, \psi_{100}(x_{100})$
- $\psi_i(x_i) = [x_i = o_i]$ with observation o_i
- Log representation: $\psi_i(x_i) = \log f_i(x_i)$

  $p(x) = \frac{1}{Z} \prod_i f_i(x_i) = \frac{1}{Z} \exp\{\sum_i \psi_i(x_i)\}$

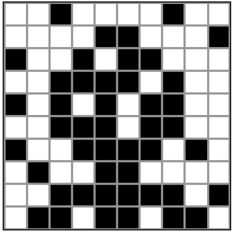


Example 2: Image Denoising

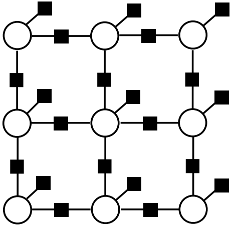


- We like to integrate prior knowledge by adding constraints to the problem
- What prior knowledge do we have about this image?
- Smoothness: Neighboring pixels tend to have the same label
- How many neighbors share the same label?
- 10 x 10 x 2 - 20 = 180 neighborhood relationships in total
- 34x label transition and 146x same label (factor 4.3 more)

Example 2: Image Denoising



$\leftrightarrow$



$$p(x_1, \ldots, x_{100}) = \frac{1}{Z} \exp \left\{ \sum_{i=1}^{100} \psi_i(x_i) + \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

- Unary log factors: $\psi_i(x_i) = [x_i = o_i]$ with observation $o_i \in \{0, 1\}$
- Pairwise log factors: $\psi_{ij}(x_i, x_j) = \lambda \cdot [x_i = x_j]$
- Parameter λ controls strength of prior $\Rightarrow$ Exercise; Next time: Stereo