Machine Learning in Graphics and Vision

Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group
MPI-IS / University of Tübingen

July 12, 2018

University of Tübingen
MPI for Intelligent Systems
Autonomous Vision Group

So far: Learning from very large Datasets

Now: Integrate Prior Knowledge

Goal

- Take probabilistic viewpoint
- Model dependency structure of problem
- Model relationships between entities parametrically
- Model complex phenomena by composing simple functions

Pros

- Easy integration of prior knowledge
- Training with limited data
- Models often easily interpretable

Cons

- Many phenomena are hard to model accurately

Overview

Structured Prediction I

- Graphical Models: Factor Graphs
- Inference: Belief Propagation

Structured Prediction II

- Stereo & Optical Flow
- Multi-view Reconstruction

Structured Prediction III

- Parameter Estimation
- Deep Structured Models

Structured Prediction

Non-Structured vs. Structured Prediction

Classification / Regression:

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

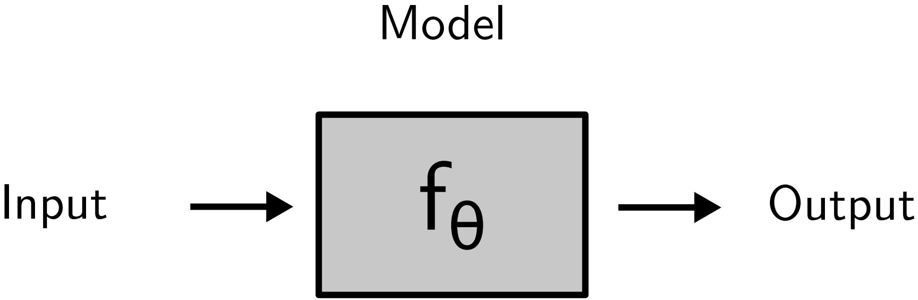
- Inputs $x \in \mathcal{X}$ can be any kind of objects
  - images, text, audio, sequence of amino acids, ...
- Output $y \in \mathbb{N} / y \in \mathbb{R}$ is a discrete or real number
  - classification, regression, density estimation, ...

Structured Output Learning:

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

- Inputs $x \in \mathcal{X}$ can be any kind of objects
- Outputs $y \in \mathcal{Y}$ 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 $f_\theta(\cdot)$

Classification / Regression

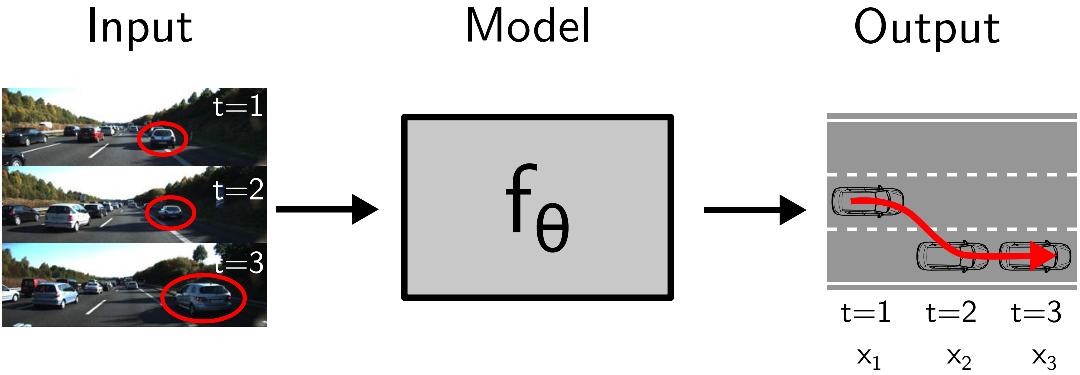
Input | Model | Output

→ f_θ → "Beach"

Classification / Regression:

- Output is encoded in a single one-dimensional variable
- Many problems are not of this form

Structured Prediction



Graphical models allow us to encode:

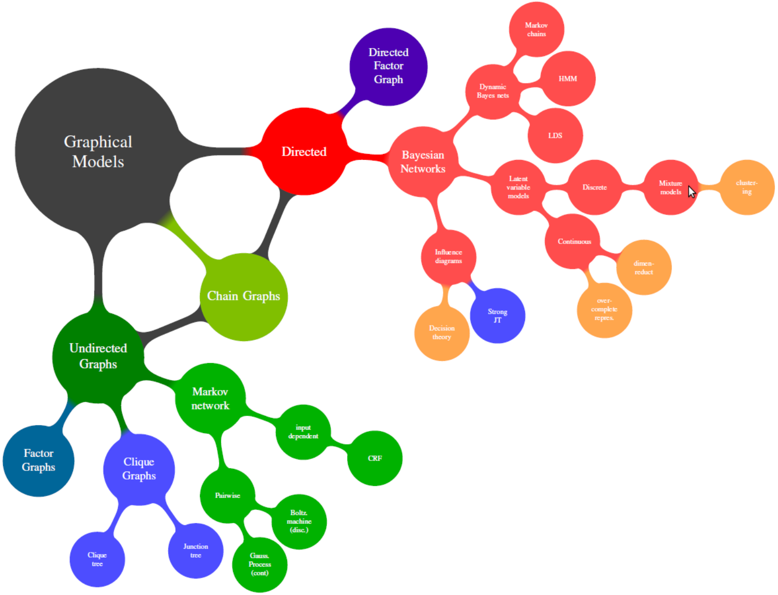
- The dependency structure of the problem
- Constraints between random variables

Probabilistic Graphical Models

Probabilistic Graphical Models

- ”Graphical language” to represent probability distributions
- In particular:
  - Dependence/independence of variables
  - Causal relationships (directed models)
- Tool to specify prior knowledge (expert knowledge)
- Allow for efficient learning & inference
- Many algorithms available
- Many different types of graphical models
  - Choice depends on application

Probabilistic Graphical Models



Markov Networks

Markov Network

Potential

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

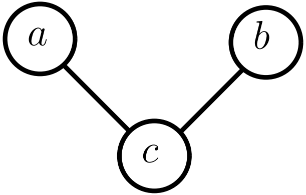
Markov Network

For a set of variables $\mathcal{X} = \{x_1, \ldots, x_D\}$ a **Markov network** is defined as a product of potentials over the **maximal cliques** $\mathcal{X}_c$ of the graph $\mathcal{G}$

$$p(x_1, \ldots, x_D) = \frac{1}{Z} \prod_{c=1}^{C} \phi_c(\mathcal{X}_c)$$

- Special case for clique size two: **Pairwise Markov network**
- If all potentials are strictly positive: **Gibbs distribution**

Example

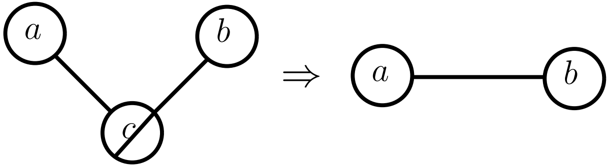


$$p(a,b,c) = \frac{1}{Z}\phi_1(a,c)\phi_2(b,c)$$

- Two maximal cliques of size two: $\phi_1(a,c)$ and $\phi_2(b,c)$
- Z normalizes the distribution and is called **partition function**

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

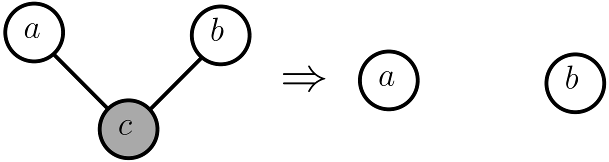
Properties of Markov Networks



- Marginalizing over c makes a and b dependent
- Proof by checking

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

Properties of Markov Networks



- Conditioning on c makes a and b independent
- Proof by checking

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

Now the general case ..

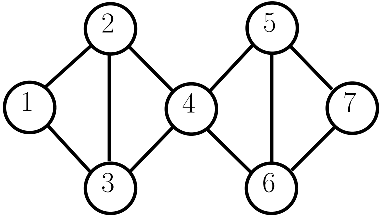
Global Markov Property

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 $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{S}$



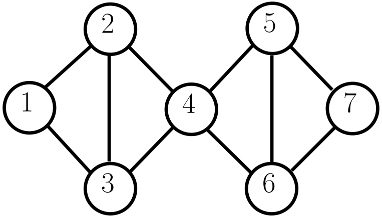
Local Markov Property

Local Markov Property

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

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

Local Markov Property – Example



- $p(x_4 \mid x_1, x_2, x_3, x_4, x_5, x_6, x_7) = p(x_4 \mid x_2, x_3, x_5, x_6)$
- In other words $x_4 \perp\!\!\!\perp \{x_1, x_7\} \mid \{x_2, x_3, x_5, x_6\}$
- And others ..

Markov Random Field (MRF)

Markov Random Field

A **Markov Random Field** is defined by a set of distributions $p(x_i \mid ne(x_i))$ with respect to an undirected graph $\mathcal{G}$ such that

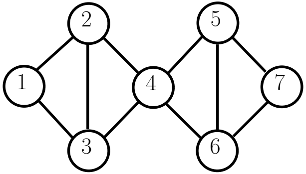
$$p(x_i \mid x_{\setminus i}) = p(x_i \mid ne(x_i))$$

- Not every set of conditional distributions $p(x_i \mid x_{\setminus i})$ yields a valid joint distribution

Finding the factorization

- An undirected graph $\mathcal{G}$ specifies a set of conditional independence statements
- Which factorization satisfies all possible independence assumptions?
- What is the most general factorization F that satisfies the independence assumptions of $\mathcal{G}$?

Finding the factorization



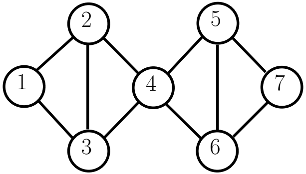
- Eliminate variable one by one
- Let's start with x_1

$$p(x_1, \ldots, x_7) = p(x_1 \mid x_2, x_3) p(x_2, \ldots, x_7)$$

since

$$p(x_1 \mid x_2, \ldots, x_7) = p(x_1 \mid x_2, x_3)$$

Finding the factorization



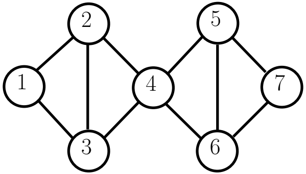
- Graph specifies:

$$\begin{aligned} & p(x_1, x_2, x_3 \mid x_4 \ldots, x_7) && = p(x_1, x_2, x_3 \mid x_4) \\ \Rightarrow \quad & p(x_2, x_3 \mid x_4, \ldots x_7) && = p(x_2, x_3 \mid x_4) \end{aligned}$$

- Hence

$$p(x_1, \ldots, x_7) = p(x_1 \mid x_2, x_3) p(x_2, x_3, \mid x_4) p(x_4, x_5, x_6, x_7)$$

Finding the factorization



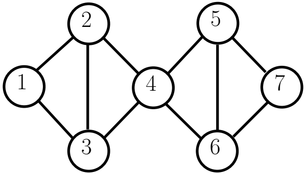
- We continue to find

$$
\begin{aligned}
p(x_1, \dots, x_7) \quad &= \quad p(x_1 \mid x_2, x_3) p(x_2, x_3 \mid x_4) \\
& \quad\quad p(x_4 \mid x_5, x_6) p(x_5, x_6 \mid x_7) p(x_7)
\end{aligned}
$$

- Factorization into clique potentials (maximal cliques)

$$
p(x_1, \dots, x_7) = \frac{1}{Z} \phi(x_1, x_2, x_3) \phi(x_2, x_3, x_4) \phi(x_4, x_5, x_6) \phi(x_5, x_6, x_7)
$$

Finding the factorization



- Markov conditions of $\mathcal{G} \Rightarrow$ factorization F into cliques
- And conversely: $F \Rightarrow \mathcal{G}$

Hammersley-Clifford Theorem

Hammersley-Clifford

Relationship Markov conditions on $\mathcal{G} \Leftrightarrow$ Factorization F holds for any undirected graph provided that the potentials are positive

- Thus also loopy ones: $x_1 - x_2 - x_3 - x_4 - x_1$
- Theorem says, distribution is of the form

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z}\phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\phi_{34}(x_3, x_4)\phi_{41}(x_4, x_1)$$

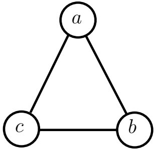
Factor Graphs

Relationship Potentials to Graphs

- Consider this factorization into potential functions:

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

- What is the corresponding Markov network (graphical model)?



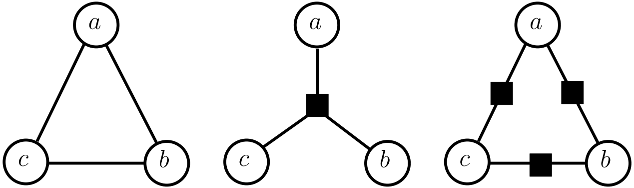
- and which other factorization is represented by this network?

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

- The factorization of the **potentials** is not uniquely specified by the graph

Relationship Potentials to Graphs

- Now we introduce an extra node (a square) for each factor



(a) (b) (c)

- Left: Markov Network
- Middle: Factor graph representation of $\phi(a, b, c)$
- Right: Factor graph representation of $\phi(a, b)\phi(b, c)\phi(c, a)$
- Different factor graphs have same Markov network (b,c)$\Rightarrow$(a)

Factor Graph Definition

Factor Graph

Given a function

$$f(x_1, \ldots, x_n) = \prod_i f_i(\mathcal{X}_i)$$

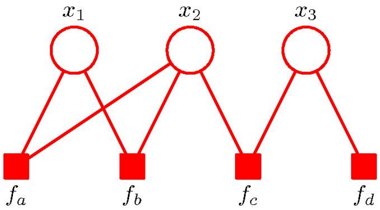
the **factor graph (FG)** has a **square node** for each factor $f_i(\mathcal{X}_i)$ and a **circle node** for each variable x_j. We typically specify this factorization up to a normalization constant

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_i f_i(\mathcal{X}_i)$$

when representing a distribution $p(\cdot)$.

Factor Graph: Example 1

- Question: which distribution ?



- Answer:

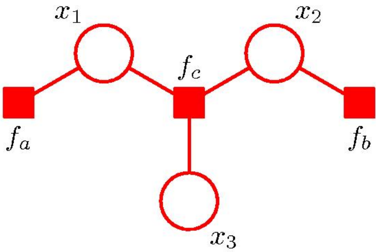
$$p(x) = \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

Factor Graph: Example 2

- Question: Which factor graph?

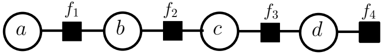
$$p(x_1, x_2, x_3) = p(x_1)\, p(x_2)\, p(x_3|x_1, x_2)$$

- Answer:



Inference in Factor Graphs

Inference in Chain Structured Factor Graphs



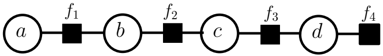
$$p(a,b,c,d) = \frac{1}{Z} f_1(a,b) f_2(b,c) f_3(c,d) f_4(d)$$

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

Computational Complexity?

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

Inference in Chain Structured Factor Graphs



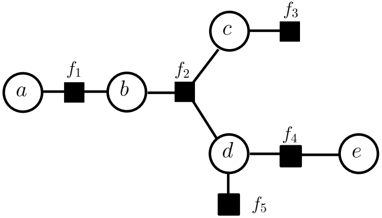
- Simply recurse further:

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

- $\mu_{m \to n}(n)$ carries the information beyond m
- Computational complexity?
- 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:



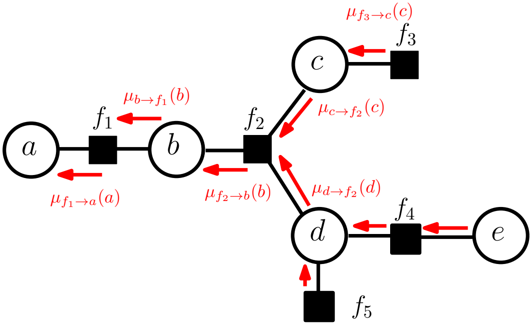
with factors

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

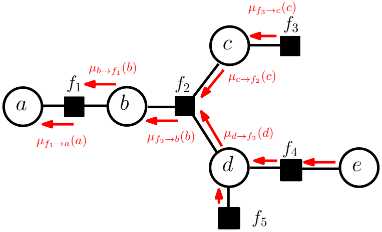
- How to find marginal $p(a,b)$?

Inference in Tree Structured Factor Graphs

- Idea: compute messages



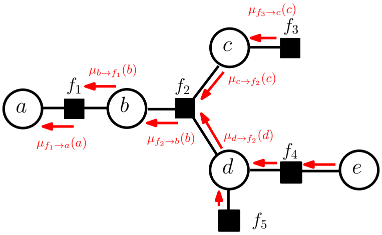
Inference in Tree Structured Factor Graphs



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

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

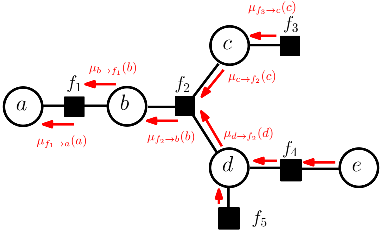
Inference in Tree Structured Factor Graphs



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

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

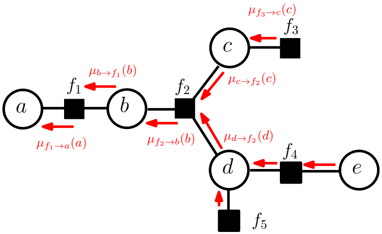
Inference in Tree Structured Factor Graphs



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

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

Factor-to-Variable Messages



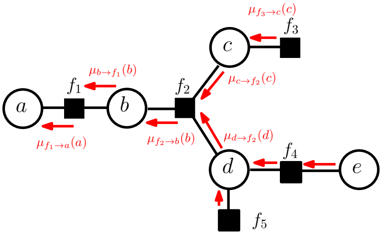
- Here (repeated from last slide):

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

- More general:

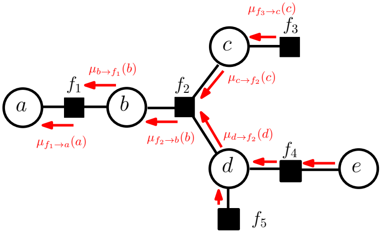
$$\mu_{f \to x}(x) = \sum_{\mathcal{X}_f \backslash x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \backslash x\}} \mu_{y \to f}(y)$$

Variable-to-Factor Messages



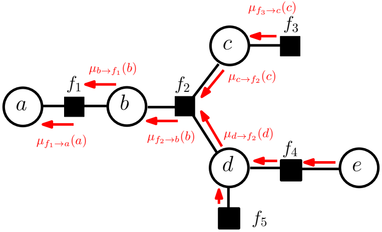
$$\mu_{d \to f_2}(d) = f_5(d) \sum_e f_4(d, e)$$

Variable-to-Factor Messages



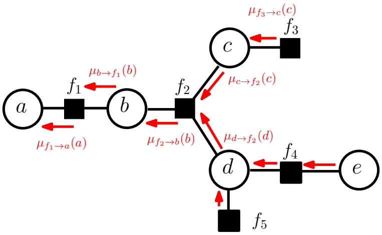
$$\mu_{d\to f_2}(d) = \underbrace{f_5(d)}_{\mu_{f_5\to d}(d)} \underbrace{\sum_e f_4(d,e)}_{\mu_{f_4\to d}(d)}$$

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:

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

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:

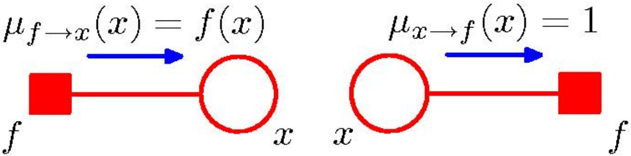
- Algorithm to compute all messages efficiently
- Assumes that the graph is singly-connected (chain, tree)

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

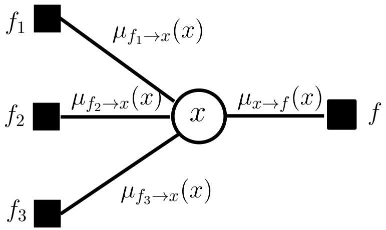
1. Initialization

- Messages from extremal node factors are initialized to factor
- Messages from extremal variable nodes can be set arbitrarily (*e.g.*, to 1)



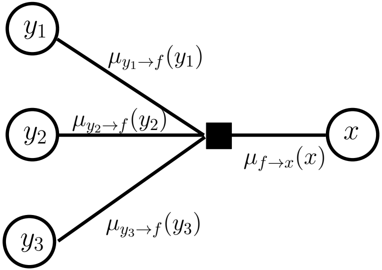
2. Variable-to-Factor Message

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



3. Factor-to-Variable Message

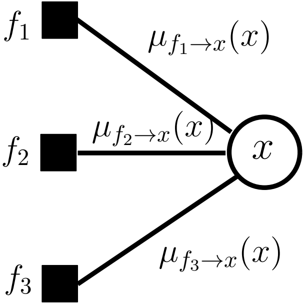
$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y\to f}(y)$$



- We sum over all states in the set of variables
- This explains the name for the algorithm (sum-product)
- Great, this is tractable now! Or not?

5. Calculate Marginals

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



Log Representation

- In large graphs, messages may become very small/big
- 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

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

then become

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

Max-Product Algorithm

Finding the maximal state: Max-Product

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

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

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

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

Example: Chain

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

- Is this what we wanted to compute in the beginning?

Example: Chain

- 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: MAP = max of ”max-marginals”

Max-Product Algorithm – Overview

Belief Propagation:

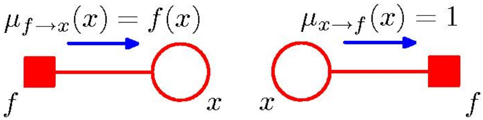
- Algorithm to compute all messages efficiently
- Assumes that the graph is singly-connected (chain, tree)

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

1. Initialisation

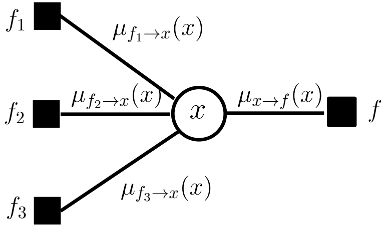
- Messages from extremal node factors are initialized to factor
- Messages from extremal variable nodes can be set arbitrarily (*e.g.*, to 1)



- Same as for sum-product

2. Variable to Factor message

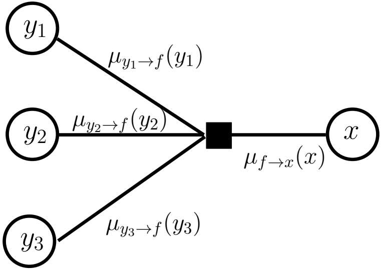
$$\mu_{x\to f}(x) = \prod_{g \in \{ne(x) \setminus f\}} \mu_{g\to x}(x)$$



- Same as for sum-product

3. Factor to Variable message

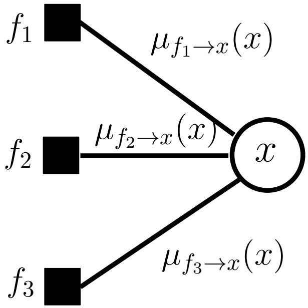
$$\mu_{f\to x}(x) = \max_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y \to f}(y)$$



- Different message than in sum-product
- This is now a max-product!

Computing the Maximal State of a Variable

$$x^* = \underset{x}{\operatorname{argmax}} \prod_{f \in ne(x)} \mu_{f \to x}(x)$$



Log Representation

- In large graphs, messages may become very small/big
- Work with log-messages instead $\lambda = \log \mu$
- Note: This doesn't change the optimization problem since

$$\log\left(\max_x p(x)\right) = \max_x \log\left(p(x)\right)$$

- Variable-to-factor messages

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

then become

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

Log Representation

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

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

then become

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

- This algorithm is called the **max-sum algorithm**

What if the graph is not singly connected?

Loopy Belief Propagation

Loopy Belief Propagation

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

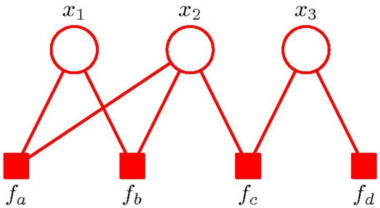
$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f\setminus x} f(\mathcal{X}_f) \prod_{y\in\{ne(f)\setminus x\}} \mu_{y\to f}(y)$$

- 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:**

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

- **Variable-to-factor messages:**

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

- $\sum_{\mathcal{X}_f\setminus x}$: 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 / factors
- To avoid large values, subtract mean from $\lambda_{x\to f}(x)$ after message update (Eq. 2)

Max-Product Belief Propagationn

- **Goal:** Find most likely state (MAP state)
- **Factor-to-variable messages:**

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

- **Variable-to-factor messages:**

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

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

Pairwise Case

Factor-to-variable messages simplify as follows.

Variable-to-factor messages don't simplify.

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

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

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

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

Pairwise Case

Factor-to-variable messages simplify as follows.

Variable-to-factor messages don't simplify.

- **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

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

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

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

Algorithm Overview

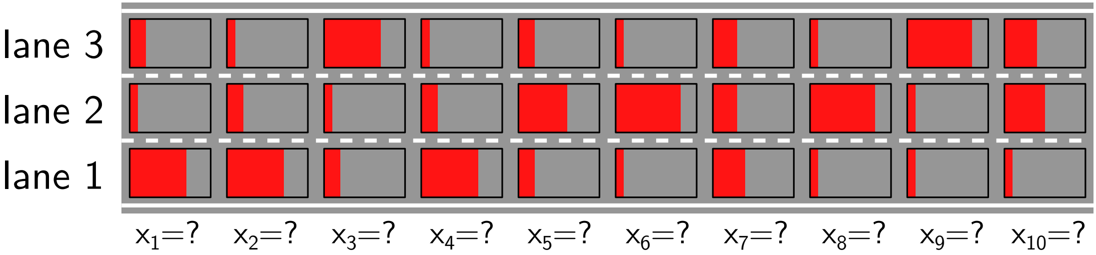
Belief Propagation Algorithm

- Input: `variables` and `factors`
- Allocate all `messages`
- Initialize the `message` log values 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`:
    $\mu_{x\to f}(x) \leftarrow \mu_{x\to f}(x) - \text{mean}\left(\mu_{x\to f}(x)\right)$
- Read off marginal or MAP state at each variable (Eq. 4 or Eq. 5)

Examples

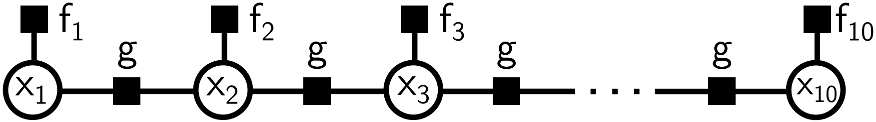
Example 1: Vehicle Localization

Example 1: Vehicle Localization



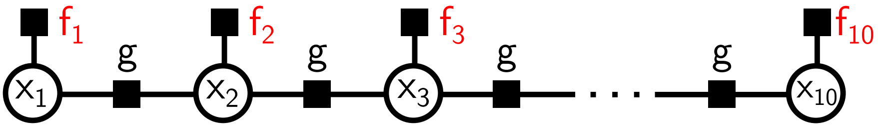
- **Goal:** Estimate vehicle location at time $t = 1, \ldots, 10$
- **Variables:** $\mathbf{x} = \{x_1, \ldots, x_{10}\} \quad x_i \in \{1, 2, 3\}$
- **Observations:** $\mathbf{o} = \{o_1, \ldots, o_{10}\} \quad o_i \in \mathbb{R}^3$

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

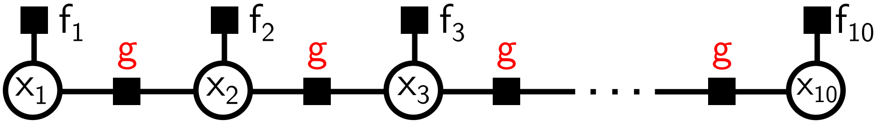


$$p_\theta(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{10} f_i(x_i) \prod_{i=1}^{9} g_\theta(x_i, x_{i+1})$$

Unary Factors:

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

Example 1: Vehicle Localization



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

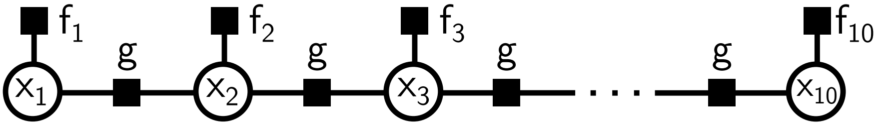
Pairwise Factors:

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

- Learning Problem:

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

Example 1: Vehicle Localization



- Maximum-A-Posteriori State:

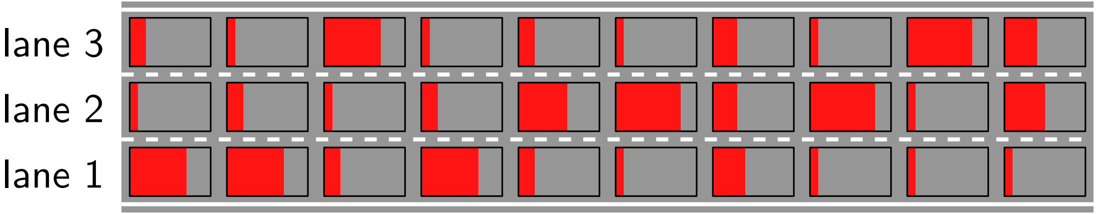
$$\hat{x}_1, \ldots, \hat{x}_{10} = \underset{x_1, \ldots, x_{10}}{\operatorname{argmax}} \, p_\theta(x_1, \ldots, x_{10})$$

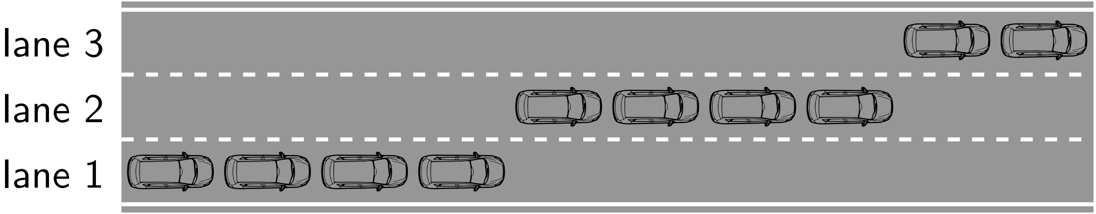
- Marginal Distribution:

$$p(x_1) = \sum_{x_2} \sum_{x_3} \cdots \sum_{x_{10}} p_\theta(x_1, \ldots, x_{10})$$

Example 1: Vehicle Localization

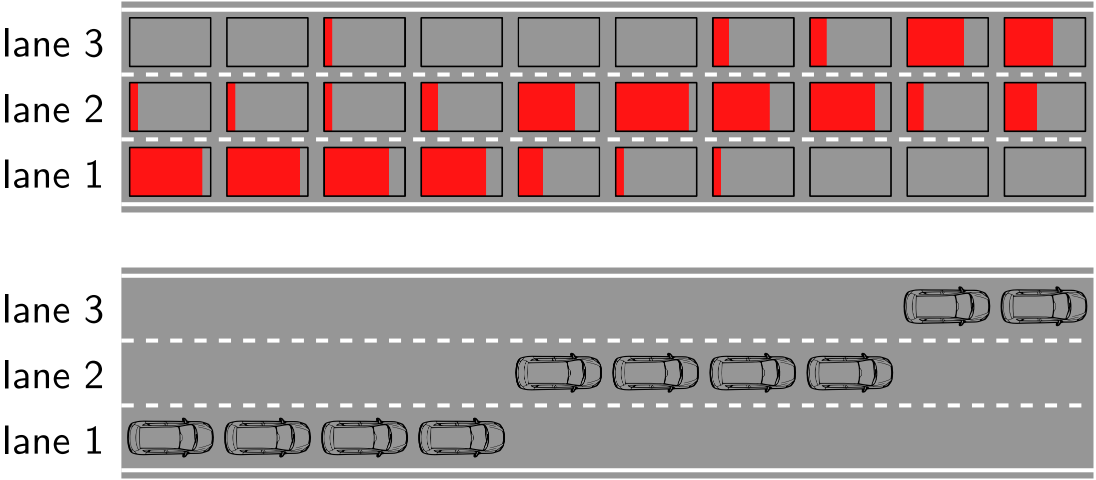
Observations





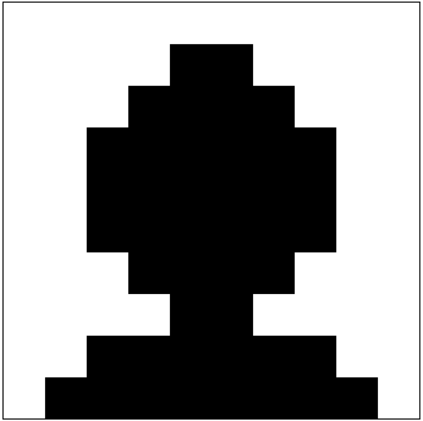
Example 1: Vehicle Localization

Marginal Distributions

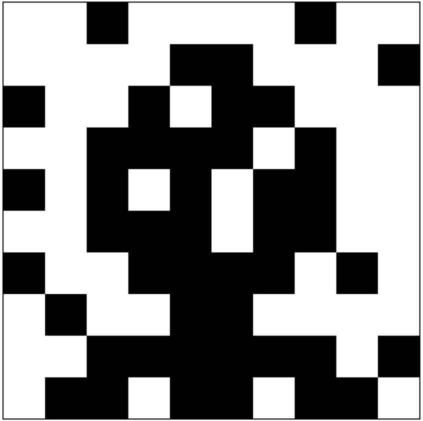


Example 2: Image Denoising

Example 2: Image Denoising

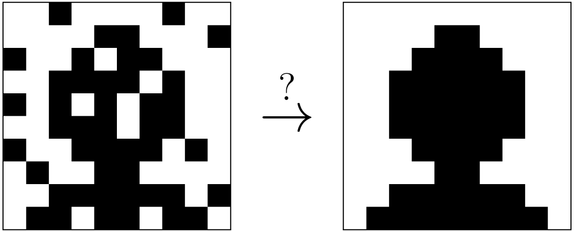


Example 2: Image Denoising

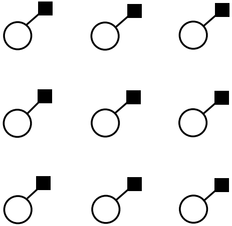


Example 2: Image Denoising

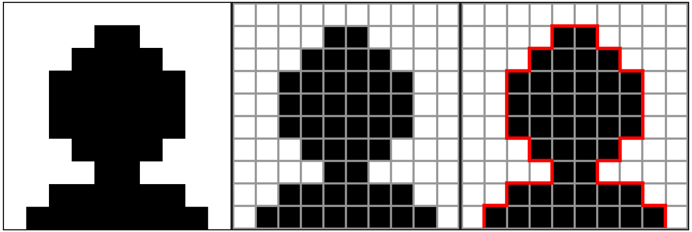
Can we recover the original image from a noisy observation?



- Variables: $x_1, \dots, x_{100} \in \{0, 1\}$
- Unary potentials: $\psi_1(x_1), \dots, \psi_{100}(x_{100})$
- $\psi_i(x_i) = [x_i = o_i]$ with observation o_i
- Log representation: $\psi_i(x_i) = \log f_i(x_i)$
  $p(x) = \frac{1}{Z} \prod_i f_i(x_i) = \frac{1}{Z} \exp\{\sum_i \psi_i(x_i)\}$

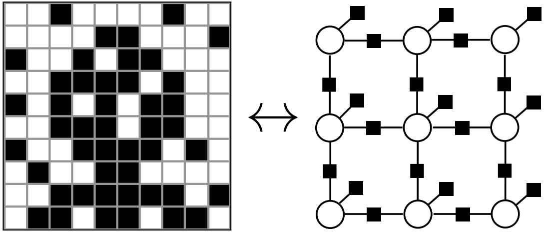


Example 2: Image Denoising



- Let us look at the clean image again!
- What prior knowledge do we have about this image?

Example 2: Image Denoising



- Log representation:

$$p(x) \propto \exp\left\{\sum_{i=1}^{100} \psi_i(x_i) + \sum_{i\sim j} \psi_{ij}(x_i, x_j)\right\}$$

- Variables: $x_1, \ldots, x_{100} \in \{0, 1\}$
- Unaries: $\psi_i(x_i) = [x_i = o_i]$ with observation $o_i \in \{0, 1\}$
- Pairwise potential: $\psi_{ij}(x_i, x_j) = \alpha \cdot [x_i = x_j]$
- Parameter α controls strength of prior

Questions?