

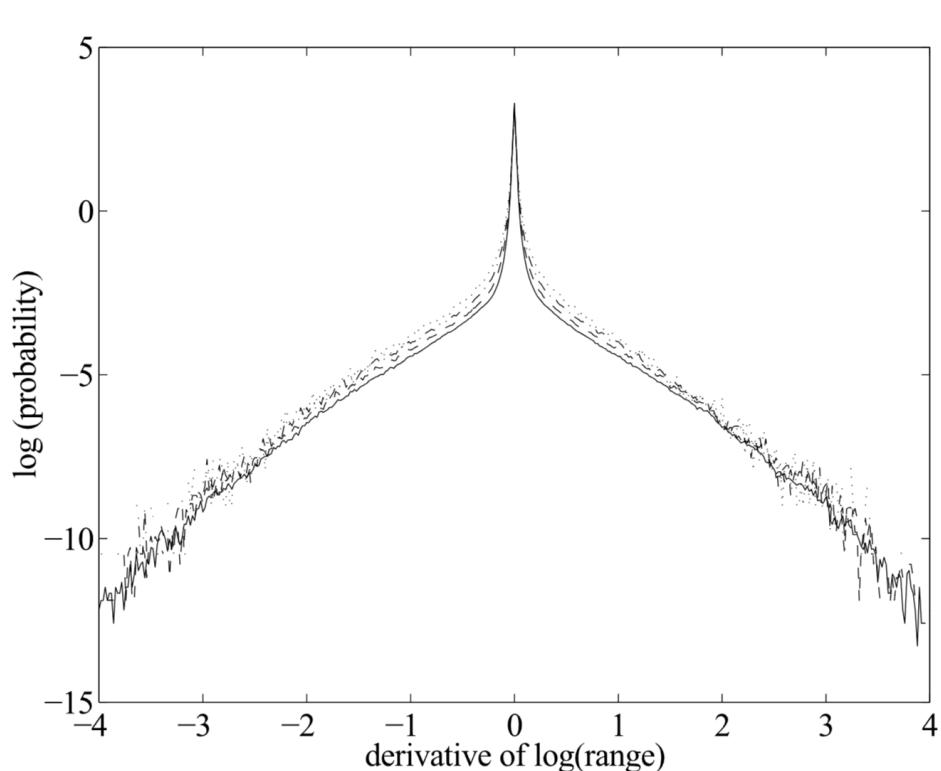
Lecture 5 - Probabilistic Graphical Models

- Lect 5 和 Lect 7 的内容是关于当今对 CV 领域仍然具有主导地位的技术，但是对于 CV 课程而言，其它过去曾流行过的技术也在我们的讨论范围内

一点题外话：从这节课开始，教授买了一些新的收音设备，音频质量可能会更好 hh

5.1 Structured Prediction

- 记得在 Lect 4 中，我们提到过关于 Stereo Matching，或者说 Block Matching 的主要问题 [Lecture 4 - Stereo Reconstruction > 4.4 Spatial Regularization](#)
 - 简单地说，Block Matching 认为两个 patch 之间的匹配应该取决于它们的相似程度——但是，当纹理很少或者重复纹理出现时，匹配就会遇到困难
 - 我们知道现实世界中，disparity 只有在物体不连续处会突变，其余位置变化速度十分缓慢



- 这里对这幅图进一步解释：横轴表示 depth 范围的导数（也即 depth 范围的变化率），纵轴表示对应的 depth 范围变化的概率

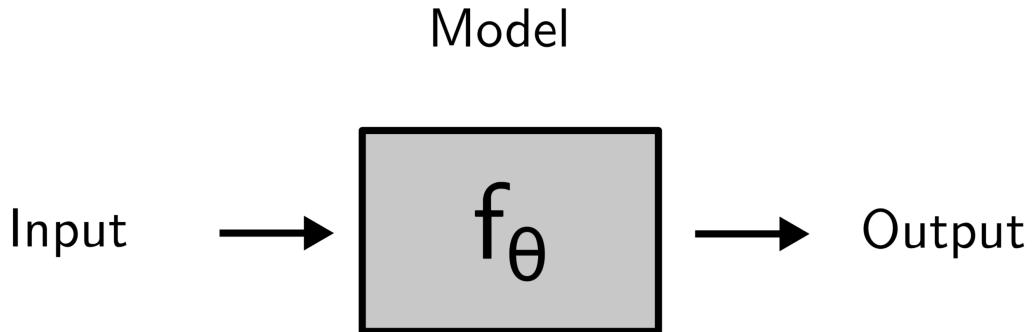
分布

- 可以理解为，一幅图中大多数的情况下，`depth`都是在缓慢变化的，少数情况下才会变化的很快
 - 利用这种先验的规律，我们想要在 Block Matching 的基础上做全局优化，这也就是 Lect 4.4 提出的 Spatial Regularization
- Spatial Regularization
 - 为了将这种先验的规律集成到 Block Matching 中，我们考虑将问题改造为一个在图像模型中 inference 的问题，其中每一个像素表示一个 node，然后我们需要根据这些 node 之间的关系来推理出整个图像模型中所有的 nodes
 -

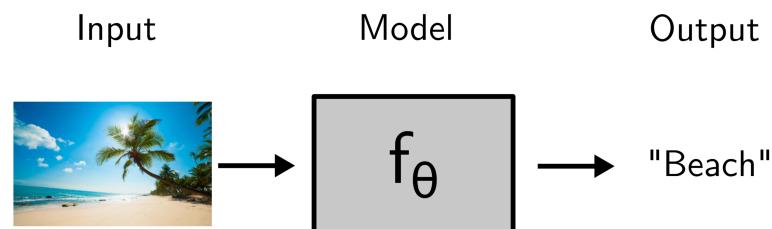
$$p(D) \propto \exp\left\{-\sum_i \psi_{data}(d_i) - \lambda \sum_{i \sim j} \psi_{smooth}(d_i, d_j)\right\}$$

- Probabilistic Graphical Models
 - 采用概率的观点，并对问题的依赖关系结构进行建模
 - 基于随机变量之间的局部约束的 Structured Prediction 模型
 - 在 DL 之前，Graphical model 主导了 CV 领域很长一段时间
 - 在训练数据很少且有先验知识可以整合时，非常有效
 - Graphical models 可以和 DL 模型结合在一起使用
 - 优点
 - 整合了先验知识
 - 使用参数少、需求数据少
 - 可以设计可解释的模型
 - 缺点
 - 很多现象的建模是十分困难的
 - 利用大型数据集比较困难（图遍历本身是 NP-hard 问题）
 - 即使对于简单的情况，推理的结果也是近似的
- 什么是 Structured Prediction 呢？
 - Classification / Regression
 - $f: X \rightarrow N$ or $f: X \rightarrow R$
 - 输入是一个高维的复杂对象（图像、文本、音频等）
 - 输出是离散/连续的数值
 - Structured Prediction

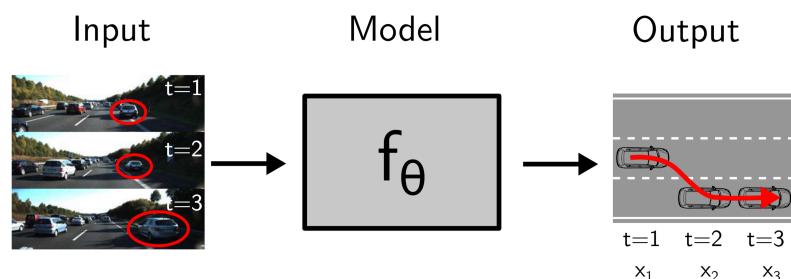
- $f : X \rightarrow Y$
 - 输入是一个高维的复杂对象 (图像、文本、音频等)
 - 输出同样是复杂的 (结构化的) 对象 (图像、文本、蛋白质结构、计算机程序等)
- Supervised Learning



- ■ Learning
 - 从训练集 $\{(x_i, y_i)\}_{i=1}^N$ 中评估参数 θ
- Inference
 - 从全新的输入 x 中预测 $\hat{y} = f_{\theta}(x)$
- Classification / Regression

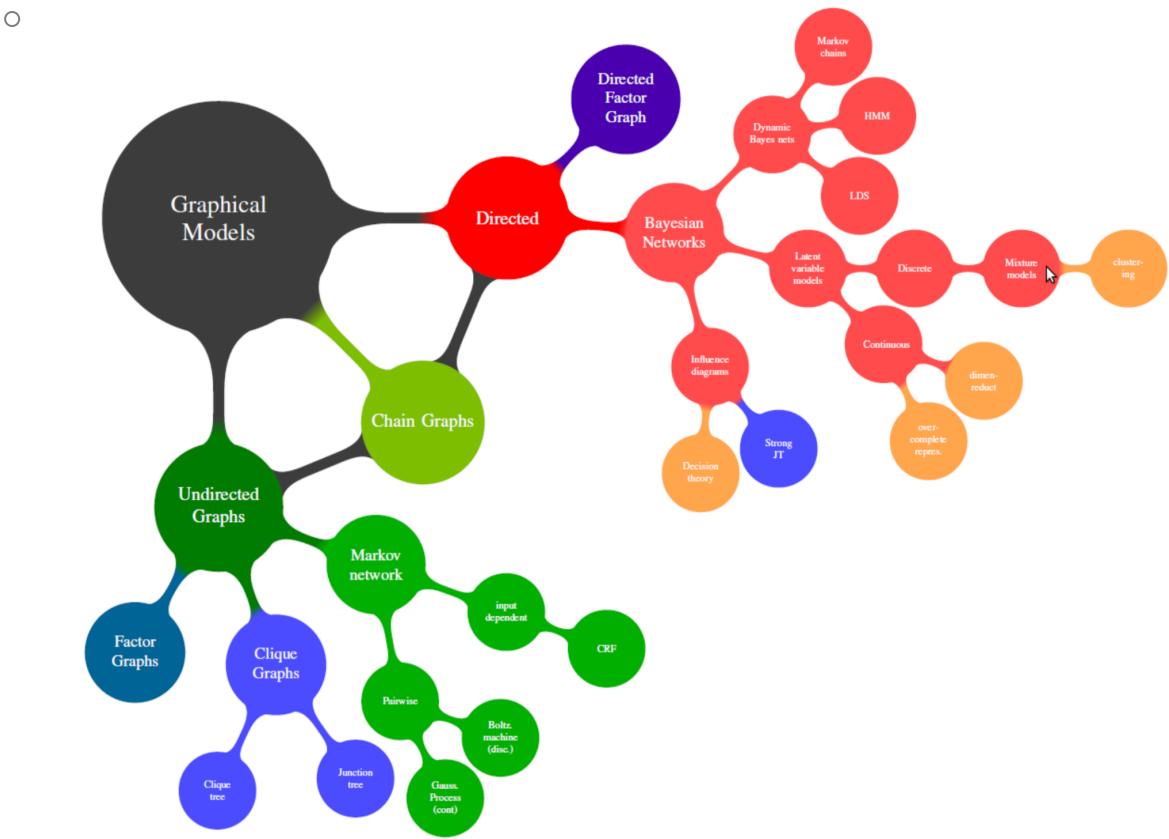


- Structured Prediction



- - 可以是Probabilistic graphical models 对问题的局部依赖关系编码
 - 或者是基于图像输出的 DL 神经网络

- Probabilistic Graphical Models



- 在本次课程中，我们只讲述其中具有代表性的一些图模型，将其分为 5、6、7 三个 Lecture 如下所示

- Lecture 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
 - [CVPR 2012 Tutorial: Structured Prediction and Learning in Computer Vision](#)
 - [Bayesian Reasoning and Machine Learning](#)

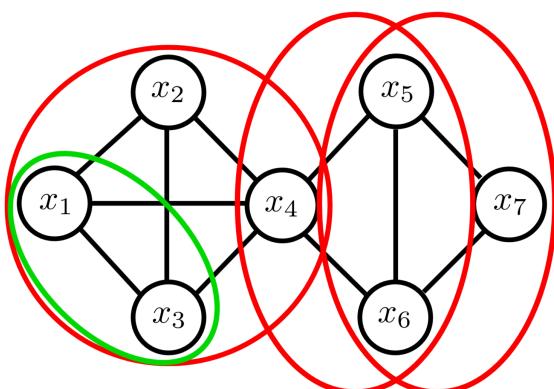
5.2 Markov Random Fields

- 概率论 Recap
 - 离散随机变量 $x \in \{1, \dots, C\}$
 - 其中随机变量 x 取到 c 的概率，表示为 $p(x = c)$
 - 连续随机变量 $x \in R$

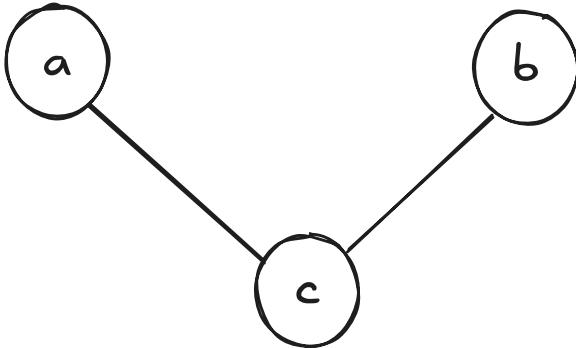
- 其中随机变量 x 在 $A(A \subset R)$ 区间内取值的概率，表示为 $p(x \in A)$
- x 上的分布，表示为 $p(x)$
- 联合分布
 - $p(x, y)$ 作为 $p(x = c, y = c')$ 的简写
- 加法规则（边缘化操作）
 - $p(x) = \sum_y p(x, y)$ 或 $p(x) = \int_y p(x, y)$
- 乘积规则
 - $p(x, y) = p(y|x)p(x)$
- Bayes 规则
 - $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$
- Markov Random Field
 - Potential (势函数)
 - potential $\phi(x)$ 被定义为变量 x 的非负函数
 - joint potential $\phi(x_1, x_2, \dots)$ 被定义为一组变量的非负函数
 - Markov Random Field (MRF) = Markov Network
 - 对于一系列变量 $\chi = \{x_1, \dots, x_M\}$, MRF 被定义为无向图 G 上所有 (maximal) cliques (极大团) $\{\chi_k\}_{k=1}^K$ 上的 potential 值的点积 \$\$
 p(\chi) = \frac{1}{Z} \prod \lim_{k=1}^K \phi(x_k)
 - 其中， $\frac{1}{Z}$ 是一个标准化参数，其中 Z 称为 * *partition function **，它确保了所有可能状态的总概率为 1。

做一下解释，Potential 本身是随机变量的一个函数，它返回随机变量的“势”，输入的随机变量的集合的势越高，说明这一组随机变量出现的概率越大

- Undirected Graph



- clique, 被定义为一组全连接的顶点
 - maximal clique, 被定义为不能再拓展更多顶点的 clique
- MRF 的属性 1/2



- \$\$

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

- 对于这样一个无向图 G , 可以找到两个极大 cliques $\chi_1 = \{a, c\}$ 、

证明过程:

- 反证法: 假设 $p(a, b) = p(a)p(b)$ 成立,

- 由于 $p(a, b)$ 等于 $p(a, b, c)$ 对 c 边缘化

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

- 又因为假设的等式成立, 所以上式 \$\$

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

$\end{aligned}$

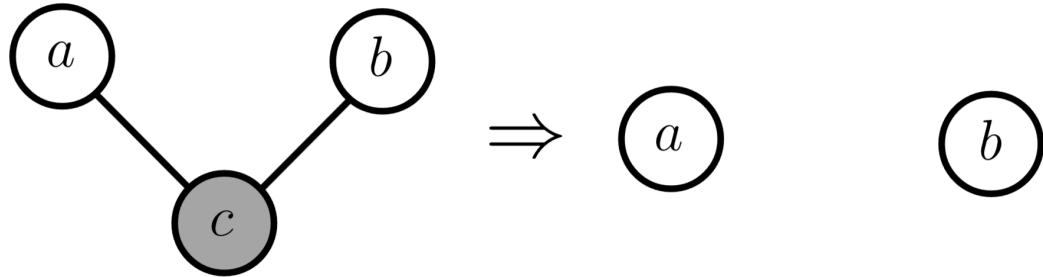
$$-\text{所以} \sum_c \phi_1(a, c) \phi_2(b, c) = \frac{1}{Z} \sum_{b,c} \phi_1(a, c) \phi_2(b, c) \sum_{a,c} \phi_1(a, c) \phi_2(b, c) - \text{又因}$$

$$\underbrace{\sum_{a,b,c} \phi_1(a, c) \phi_2(b, c)}_{\underbrace{\sum_c \phi_1(a, c) \phi_2(b, c)}_{\{2\}}}$$

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

—所以证明， $p(a, b) \neq p(a)p(b)$ 在任意条件下成立是错误的

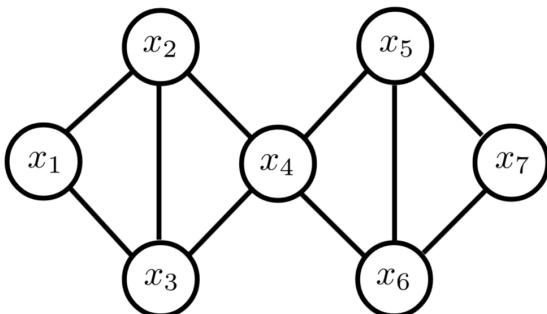
- MRF 的属性 2/2



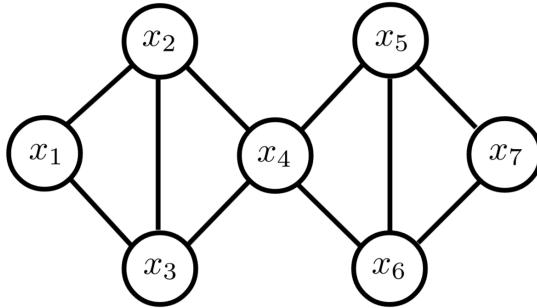
- 如果我们 conditioning 节点 c , 会使得节点 a 和 b 独立
 - 我们将这一过程写为 $a \perp\!\!\!\perp b | c$
 - 可以由 $p(a, b | c) = p(a | c)p(b | c)$ 证明得到

上面的属性可以推广得到下面的性质

- 全局 MRF 性质
 - 如果所有的从 A 中出发的节点都要经过子集 S 才能到达 B , 我们称子集 S 将 A 和 B 分开
 - 性质
 - 对于满足上述条件的集合 (A, B, S) , 其中 S 分开 A 和 B , 我们可以得到 $A \perp\!\!\!\perp B | S$ (A 于条件 S 独立于 B)



- 对于上图的情况， x_4 可以作为左右两侧集合的 S 集合；也可以令
 $A = \{x_1\}$ 、 $B = \{x_7\}$ ，则其余的部分作为 S 集合，条件也成立
- 局部 MRF 性质
 - 从全局 MRF 性质，我们可以很容易推导出局部 MRF 性质
 - 性质
 - 当一个节点 x 被它的相邻节点 condition 时，节点 x 会与图中剩下的所有节点相独立
 - $p(x|\chi/\{x\}) = p(x|ne(x))$
 - 其中相邻节点的集合 $ne(x)$ 称为 **Markov blanket**
 - 这条性质当节点 x 是一个节点集合时，也成立

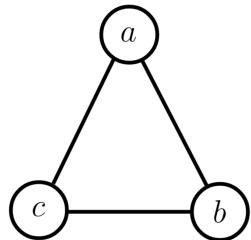


- ■ 同理，在这个例子中，我们可以得知
 $p(x_4|x_1, x_2, x_3, x_5, x_6, x_7) = p(x_4|x_2, x_3, x_5, x_6)$
 - 换句话说， $x_4 \perp\!\!\!\perp \{x_1, x_7\} | \{x_2, x_3, x_5, x_6\}$
- Hammersley-Clifford 理论 (★)
 - 当且仅当无向图 G 是 Gibbs 随机场时，具有严格正质量或密度的概率分布满足 **Markov 性质**，也即它可以在无向图的极大 Cliques 上分解为 potential 函数乘积的形式
- Filter View of Graphical Models
 - 只有那些根据极大 clique 分解的概率分布，才符合图模型的要求
 - 或者，可以说只有满足 Markov 性质的分布才符合要求
 - 根据 Hammersley-Clifford 理论，我们可以知道上述两种说法是等价的（满足其一即可）

上面的条件给图模型定义了一个“过滤器”，只有满足这些条件的图模型才可以“通过”

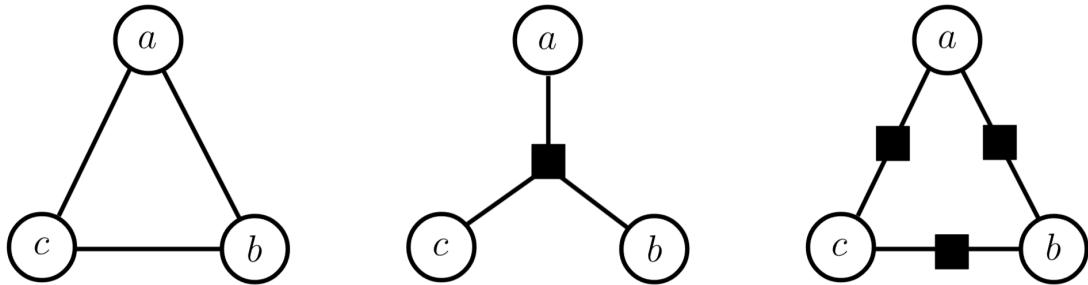
5.3 Factor Graphs

- Factor Graph 是一种比 MRF 更简单的模型，为什么我们需要 Factor Graph？
 - 考虑下面的分解情况
 - $p(a, b, c) = \frac{1}{Z} \phi(a, b) \phi(b, c) \phi(c, a)$
 - 很显然， (a, b, c) 是一个极大 clique，那么这个 Markov Net 对应的图应该这样表示



- ◦ 似乎根据这个图，我们也可以写出这样的等式

$$p(a, b, c) = \frac{1}{Z} \phi(a, b, c)$$
- 由此我们发现，Markov 的图可能同时对应多个分解式——但是这两个分解式并非等价，第二个分解式可以代表更大范围的分布类型；但是这两个都遵循 Markov 性质
- 因此，对于 Markov 图，这种因式分解方式并非唯一，我们需要找到一种新的区分方式



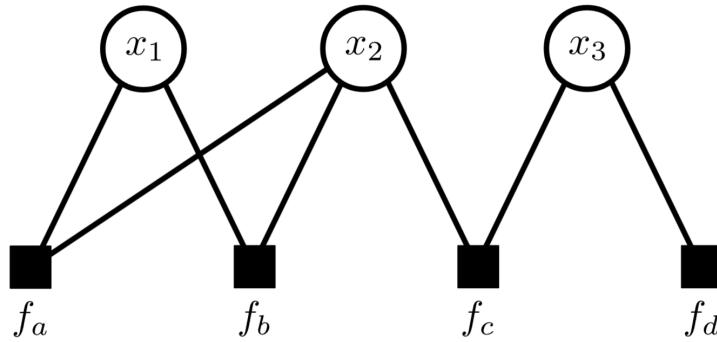
- ▪ 我们用一种额外的节点，方块节点，来表示 factor
 - 左侧的图：表示传统的 Markov Network
 - 中间的图：表示 Factor Graph $\frac{1}{Z} \phi(a, b, c)$
 - 右侧的图：表示 Factor Graph $\frac{1}{Z} \phi(a, b, c)$
- 两种 Factor Graph 指向了同一种 Markov Network，我们通过这种方式来区分同一种 Markov Network 的不同分解方式
- Factor Graph
 - 令 $\chi = \{x_1, \dots, x_M\}$, $\{\chi_k\}_{K=1}^K$, 其中 $\chi_K \subseteq \chi$
 - 给出函数 $f(\chi) = \prod_{k=1}^K f_k(\chi_k)$

- Factor Graph (FG) 是一个二部图，对每一个 factor f_k 有一个方块节点，对每一个随机变量 x_i 有一个圆形节点——通过标准化 f 函数，我们可以得到分布 $p(\chi) = \frac{1}{Z} \prod \text{limits}^{\{K\}}_{k=1} f_k(\chi_k)$

> 实际上和 *MarkovGraph* 的定义很像，只是将 $p(x)$ 换成了 f

二部图中，同一形状的节点之间不能连接

- example1

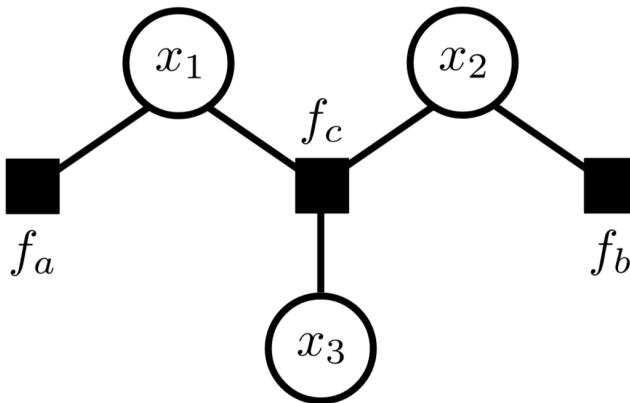


- 给出一个 FG，我们可以很容易的写下它所对应的唯一的分解式
 - $p(\chi) = \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$

- example2

- 对应的，如果有给出的分解式 $p(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3|x_1, x_2)$ ，我们也可以很容易画出它对应的结构唯一的 FG
 - 这里的每一个 p 都对应一个独立的 f ，因此我们画出的图应该是这样的

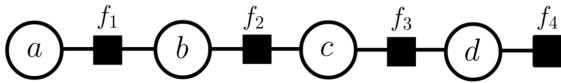
■



5.4 Belief Propagation

- Belief Propagation 是一种在 FG 模型中进行推理的算法

- 首先我们考虑一下链式的 FG 的推理过程



- 如图, 它的分解式是 $p(a, b, c, d) = f_1(a, b)f_2(b, c)f_3(c, d)f_4(d)$
 - 我们想要求解 $p(a)$ 时, 根据上一小节的知识, 我们可以将 a 边缘化, 得到 $p(a) = \sum_{b,c,d} p(a, b, c, d)$ 的式子来计算
 - 如果每一个随机变量都有 k 种取值可能, 那么这里的计算复杂度就有 k^3 , 假设最简单的情况, $k = 2$, 这里这里也就是 $2^3 = 8$, 可见它会随着链的长度而呈现指数式的增长
 - 因此, 我们可以用一些技巧来降低计算某个节点概率的复杂度
 - 如果我们想计算 $p(a, b, c)$, 则需要边缘化 d , 也即

$$p(a, b, c) = \sum_d p(a, b, c, d) = \frac{1}{Z} \sum_d f_1(a, b)f_2(b, c)f_3(c, d)f_4(d)$$
 - 其中, 只有后两项是与 d 相关的, 因此我们可以把前面提出来

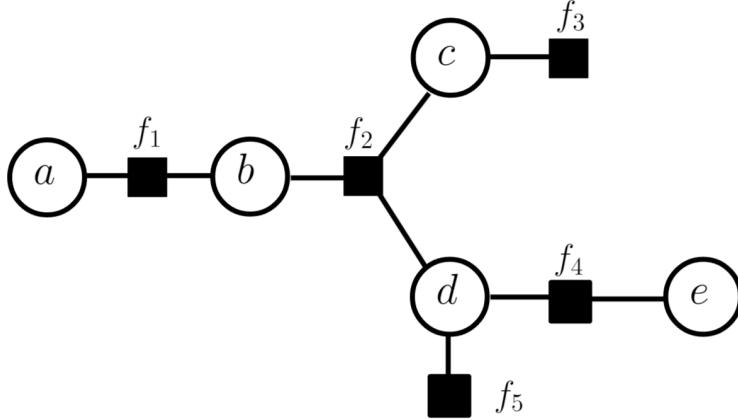
$$p(a, b, c) = \frac{1}{Z} f_1(a, b) f_2(b, c) \underbrace{\sum_d f_3(c, d) f_4(d)}_{\mu_{d \rightarrow c}(c)}$$
 - 此时, 大括号部分的表达式对 d 边缘化, 也即它只是关于 c 的表达式, 只有 c 是自由变量——我们将这个等式称为一条 message, 由变量 d 发送到变量 c 的消息, 这个消息的取值只取决于变量 c , 所以 $\mu_{d \rightarrow c}$ 表示这是一条由 d 到 c 的消息, (c) 表示这个消息的参数是 c
 - 那么同理, 我们想计算 $p(a, b)$ 时, 就需要计算

$$p(a, b) = \sum_c p(a, b, c) = \frac{1}{Z} f_1(a, b) \underbrace{\sum_c f_2(b, c) \mu_{d \rightarrow c}(c)}_{\mu_{c \rightarrow b}(b)}$$
 - 想计算 $p(a)$ 时, 只需要计算

$$\sum_b p(a, b) = \frac{1}{Z} \sum_b f_1(a, b) \mu_{c \rightarrow b}(b) = \frac{1}{Z} \mu_{b \rightarrow a}(a)$$
 - 综上所述, 我们采用了一种递归的想法, 每一次迭代需要计算两项的乘积, 也即假设链中有 M 个随机变量, 则它的复杂度为 $(M - 1) \times 2$ (如果随机变量只有 2 种取值的话)

事实上, 在链式结构中, 引入 factor 的作用无法体现出来, 我们需要看一些更复杂一点的结构

- 对于树形结构的 FG 推理过程

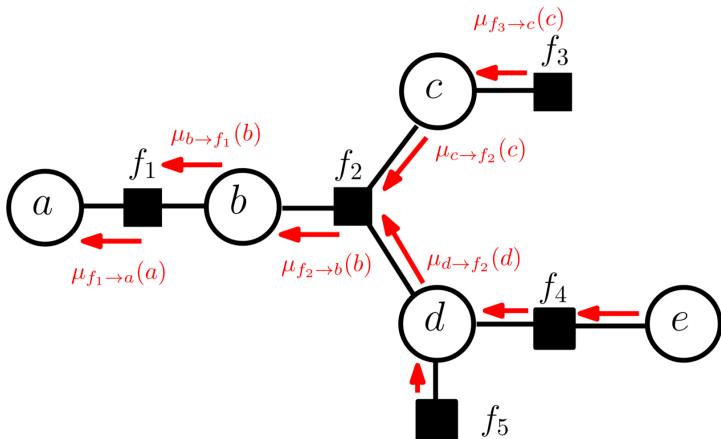


- 我们可以写出它的分解式为

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

- 我们该如何计算 $p(a, b)$?

- 根据之前的 message 的概念，我们可以写出每一条边的 message 表达式



- 然后我们可以一步步写

$$- p(a, b) = \frac{1}{Z} f_1(a, b) \underbrace{\sum_{c,d,e} f_2(b, c, d) f_3(c) f_4(d, e) f_5(d)}_{\mu_{f2->b}(b)}$$

$$\mu_{f2->b}(b) = \sum_{c,d} f_2(b, c, d) f_3(c) f_5(d) \sum_e f_4(d, e) = \sum_{c,d} f_2(b, c, d) \underbrace{f_3(c) f_5(d)}_{\mu_{c->f2}(c)} \underbrace{\sum_e f_4(d, e)}_{\mu_{d->f2}(d)}$$

$$- \mu_{f2->b}(b) = \sum_{c,d} f_2(b, c, d) \mu_{c->f2}(c) \mu_{d->f2}(d)$$

- 做到现在，我们可以用一个统一的方式来表达从 factor 到 variable 的 message 的生成

```

$$
\left\{
\begin{aligned}
& \mu_{f \rightarrow x}(x) = f(x) \quad \text{\& leaf factors} \\
& \mu_{f \rightarrow x}(x) = \sum \lim_{\chi(f)} \backslash \mu_{\chi(f)} \backslash \backslash \\
& x f(\chi(f)) \prod \lim_{y \in ne(f) \backslash x} \mu_{y \rightarrow f} \\
& (y) \quad \text{\& otherwise} \\
\end{aligned}
\right.

```

—其次， $\mu_{d \rightarrow f_2}(d) = \underbrace{f_5(d)}_{\mu_{f_5 \rightarrow d}(d)} \underbrace{\sum_e f_4(d, e)}_{\mu_{f_4 \rightarrow d}(d)}$ — 我们也可以用一个统一的方式表达

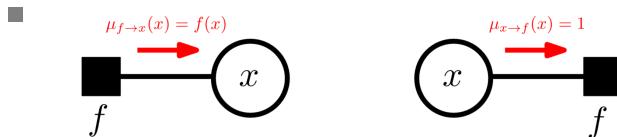
$$\mu_x(x \rightarrow f) = \prod \lim_{g \in ne(x) \backslash f} \mu_g(g \rightarrow x)$$

> 注意，这里的 $ne(f)$ 、 $ne(x)$ 都表示“incoming”的相邻 f 节点，也即输入

- 现在，我们已经明白了，整个传播过程其实都可以用消息的组合方法来表示，并且可以通过传播的方式进行推理，下面我们将介绍几种利用这种思想的算法，用于有效推理 FG

5.4.1 Sum-Product Algorithm

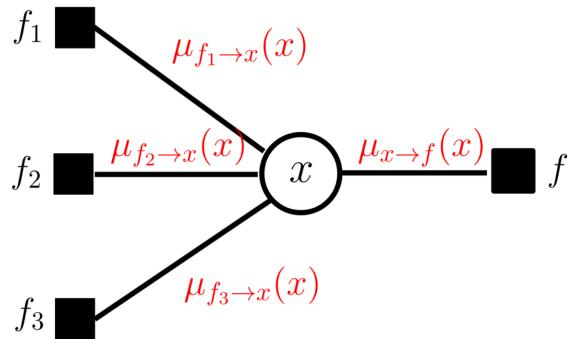
- Sum-Product 算法，也称为 Belief Propagation
 - 高效计算所有的 message
 - 假设图时单向连接的（链、树等，也即没有环结构）
- 算法步骤
 1. 初始化（处理所有的叶子节点）
 - 对于 factor 的叶子节点，其 message 为 $f(X)$
 - 对于 variable 的叶子节点，其 message 为 1



2. 计算所有的 variable to factor 的 message

○

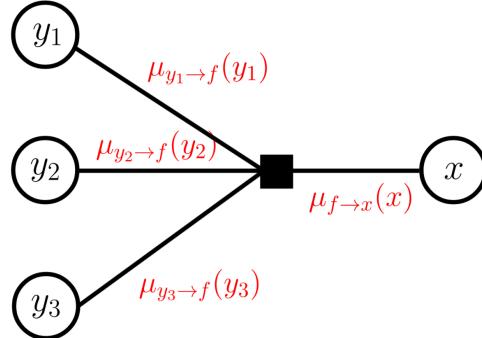
$$\mu_{x \rightarrow f}(x) = \prod_{g \in \{ne(x)\} \setminus f} \mu_{g \rightarrow x}(x)$$



3. 计算所有的 factor to variable 的 message

○

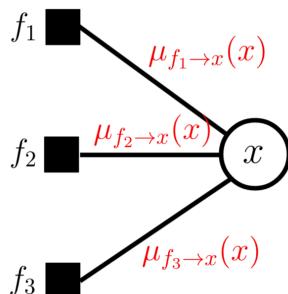
$$\mu_{f \rightarrow x}(x) = \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f)\} \setminus x} \mu_{y \rightarrow f}(y)$$



4. 重复 2、3 步骤，直到所有的 message 都被计算完毕并记录

5. 从 messages 中计算需要的边缘

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



○

- 与步骤 2 不同，这里的节点（集） x 没有要发送消息的 factor 节点，因此我们只需要将所有输入它的消息相乘
- 我们知道 $p(x) \propto \prod_{f \in ne(x)} \mu_{f \rightarrow x}(x)$

- Log Representation

- 由于我们的 message 计算过程涉及到非常多的连乘，当 FG 非常大时，中间可能会出现非常大或非常接近于 0 的数，而计算机的浮点精度是有限的——所以，这种情况下我们会使用 log 表示法

■ 令 $\lambda = \log \mu$

○ 则 factor to variable

$$\circ \quad \mu_{x \rightarrow f}(x) = \prod_{g \in \{ne(x) \setminus f\}} \mu_{g \rightarrow x}(x)$$

○ 变为

$$\circ \quad \lambda_{x \rightarrow f}(x) = \sum_{g \in \{ne(x) \setminus f\}} \lambda_{g \rightarrow x}(x)$$

○ variable to factor

$$\circ \quad \mu_{f \rightarrow x}(x) = \sum_{\chi_f \setminus x} f(\chi_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y \rightarrow f}(y)$$

○ 变为

$$\circ \quad \lambda_{f \rightarrow x}(x) = \log \left(\sum_{\chi_f \setminus x} f(\chi_f) \exp \left[\sum_{y \in \{ne(f) \setminus x\}} \lambda_{y \rightarrow f}(y) \right] \right)$$

○ 也即，我们将原来的连乘符号都等价替换为了连加符号

5.4.2 Max-Product Algorithm

- 还有一种算法称为 Max-Product 算法，非常类似，但是要解决的问题目标不同
 - 对于给定的分布 $p(a, b, c, d)$ ，我们想要找到一组 a^*, b^*, c^*, d^* ，使得可以令 $p(a, b, c, d)$ 最大化
 - 这个问题称为 Maximum-A-Posteriori(MAP) 问题
- 我们以 $p(a, b, c, d) = \frac{1}{Z} f_1(a, b) f_2(b, c) f_3(c, d)$ 为例

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

- 目前我们计算出了最大的概率值，但是没有给出最大时， a, b, c, d 的状态
 - 因此，当 message 被计算好，我们用回溯的方式来计算对应的 a, b, c, d 的值
 - \$\$

```
\begin{aligned}
a^ &= \arg\max_a, \mu{b \rightarrow a}(a) \\
b^ &= \arg\max_b, f_1(a^*, b), \mu{c \rightarrow b}(b) \\
c^ &= \arg\max_c, f_2(b^*, c), \mu{d \rightarrow c}(c) \\
d^ &= \arg\max_d, f_3(c^*, d)
\end{aligned}
```

\$\$

- 这也被称为 ****backtracking**** (DP 算法)，或者也被称为 **Viterbi** 算法
- 当 MAP 问题的解唯一时，我们将其称为 “**max-marginals**”，此时的解很容易计算——不涉及到多目标优化 **Pareto** 前沿的问题

5.4.3 Loopy Belief Propagation

- 然而，在 CV 中，有不少的情况都涉及到带有循环结构的图

$$\begin{aligned} \mu_{x \rightarrow f}(x) &= \prod_{g \in \{ne(x)\} \setminus f} \mu_{g \rightarrow x}(x) \\ \mu_{f \rightarrow x}(x) &= \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f)\} \setminus x} \mu_{y \rightarrow f}(y) \end{aligned}$$

○

- 但是这些 message 的计算公式仍然适用
- 因此，我们也可以把这些算法轻松应用在循环结构的图中
- 只是，结果不再是精准的，而是一个近似的大约值
- 甚至，最后的结果不一定是收敛的 [Yedida et al. 2004]
- 但是在 CV 应用中，结果往往出奇的好

循环结构的问题是什么？——到达目标节点的路不再唯一了，也即可能有很多个 message 的过程都可以达到我们的目标节点

- 那么，哪些 message 会被使用到呢？

- 随机选择，或提前设好固定的顺序
- 比较流行的选择
 1. factor -> variable
 2. variable -> factor
 3. 重复 N 轮迭代
- 当 FG 是二部图时，我们也可以考虑并行进行（实际上，只要是多叉的，我们都可以考虑并行代码）

5.4.4 Summary

这部分主要用于 exercise 的提示，以及总结前面的内容，方便总结及快速查阅

● Sum-Product Belief Propagation

- Goal: Compute **marginals** of distribution
 - ▶ Factor-to-variable messages:

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

- ▶ Variable-to-factor messages:

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

- ▶ $\sum_{\mathcal{X}_f \setminus x}$: 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 $\lambda_{x \rightarrow f}(x)$ after message update (Eq. 2)

● Max-Product Belief Propagation

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

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

- ▶ Variable-to-factor messages:

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

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

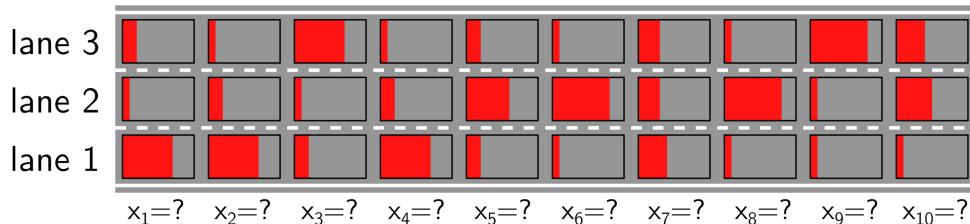
● Algorithm Overview

- **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:

$$\lambda_{x \rightarrow f}(x) = \lambda_{x \rightarrow f}(x) - \text{mean}(\lambda_{x \rightarrow f}(x))$$
 - ▶ Read off marginal or MAP state at each variable (Eq. 4 or Eq. 5)

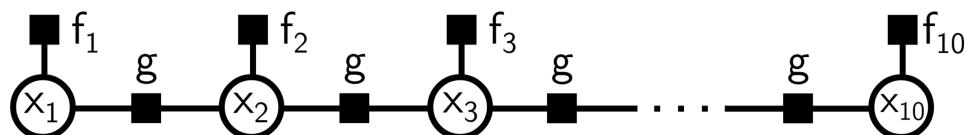
5.5 Examples

5.5.1 Ex1. Vehicle Localization



- 考虑这样一个问题——有一辆车，10个 time step，3条车道线；车在任意一个 time step 下可能在任一条车道线上
 - 目标：预测车在每一个 time step 下的位置
 - 随机变量： $\chi = \{x_1, \dots, x_{10}\}$ $x_i \in \{1, 2, 3\}$
 - 观测值（可以是由预测模型生成的）：

$$O = \{o_1, \dots, o_{10}\} \quad o_i \in R^3$$
- 我们可以构建如下的 FG



- 其中 $f_1 \sim f_{10}$ 表示 10 个观测值，每两个随机变量之间有一个相同的 factor g
- 分解式 $p_\theta(x) = \frac{1}{Z} \prod_{i=1}^{10} f_i(x_i) \prod_{i=1}^9 g_\theta(x_i, x_{i+1})$
- 其中观测值的例子如下



Unary Factors:

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

- factor g 的例子如下

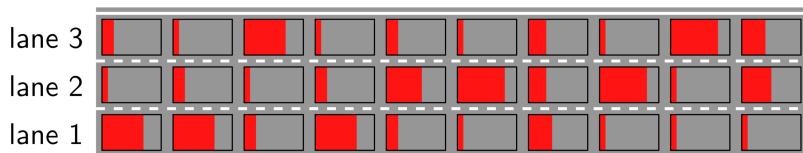
Pairwise Factors:

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

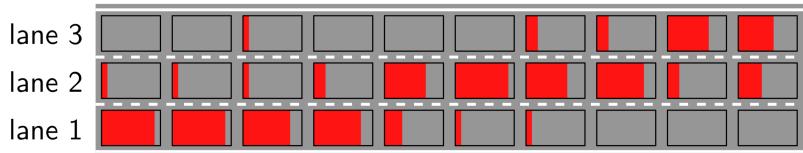
- ○ 这里我们根据先验知识: Lane1 不可能直接跳到 Lane3、Lane3 也不可能直接跳到 Lane1，设定 Pairwise 的 factor
- 推理完成后，我们来看看观测值和推理后的效果区别



Observations



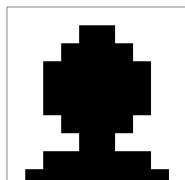
Marginal Distributions



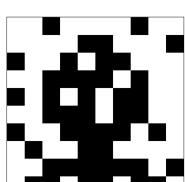
- 可以看到，那些违背我们先验知识的情况已经被修正

5.5.2 Ex2. Image Denoising

- 我们有相机拍摄的图片，正常情况下应该是这样的



- 但是相机出了点问题，变成了这样



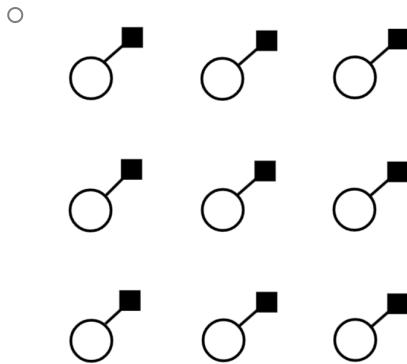
- 我们想要将带有噪声的图片修正为正常的图片

- 变量: $x_1, \dots, x_{100} \in \{0, 1\}$ (二值化像素)

- 定义一元 log factor: $\psi_1(x_1), \dots, \psi_{100}(x_{100})$

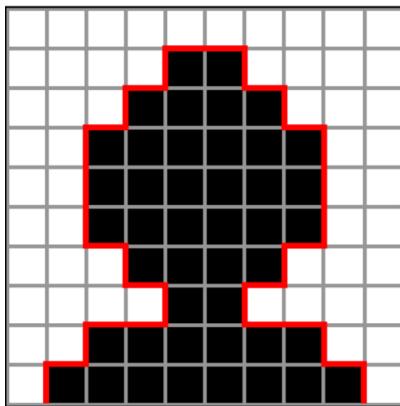
- $\psi_i(x_i = [x_i = o_i])$ (我们先简单定义，当 $x_i = o_i$ 时，置信度为 1)

- 则 FG 如下

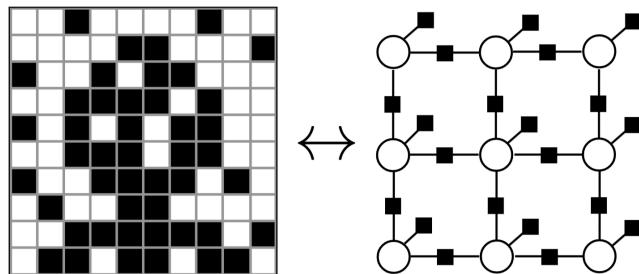


- 分解式: $p(x) = \frac{1}{Z} \prod_i f_i(x_i) = \frac{1}{Z} \exp[\sum_i \psi_i(x_i)]$

- 此时如果我们推理，会发现图像一点都没有变化——因为我们还没有引入先验知识



- - 对于这幅图，有 44 个黑色像素，56 个白色像素
 - 我们考虑所有的边 (因为边才是决定 message 的因素)，有 $10 \times 10 \times 2 - 20 = 180$ 个边，其中有 34 条边是黑白转换，146 条边是同样的颜色
 - 因此，一块像素保持颜色与转换颜色的概率比为 146:34



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

- Unary log factors: $\psi_i(x_i) = [x_i = o_i]$ with observation $o_i \in \{0, 1\}$
- Pairwise log factors: $\psi_{ij}(x_i, x_j) = \lambda \cdot [x_i = x_j]$

- ■ 据此，我们可以引入 Pairwise factor，其中 λ 为超参数，也即我们认为两块像素相等的概率有多少