Lecture 5: Graphical Models: Inference

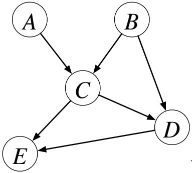
4F13: Machine Learning

Zoubin Ghahramani and Carl Edward Rasmussen

Department of Engineering, University of Cambridge

January 28th, 2010

Inference in a graphical model

Consider the following graph: 

which represents:

$$p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|B, C)p(E|C, D)$$

Inference: evaluate the probability distribution over some set of variables, given the values of another set of variables.

For example, how can we compute $P(A|C = c)$? Assume each variable is binary.

Naive method:

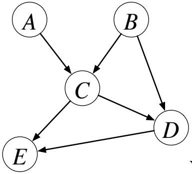
$$p(A, C = c) = \sum_{B,D,E} p(A, B, C = c, D, E) \quad \text{[16 terms]}$$

$$p(C = c) = \sum_{A} p(A, C = c) \quad \text{[2 terms]}$$

$$p(A|C = c) = \frac{p(A, C = c)}{p(C = c)} \quad \text{[2 terms]}$$

Total: 16+2+2 = 20 terms have to be computed and summed

Inference in a graphical model

Consider the following graph:  which represents:

$$p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|B, C)p(E|C, D)$$

Computing $p(A|C = c)$.

More efficient method:

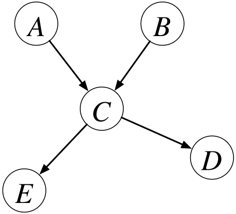
$$\begin{aligned} p(A, C = c) &= \sum_{B,D,E} p(A)p(B)p(C = c|A, B)p(D|B, C = c)p(E|C = c, D) \\ &= \sum_{B} p(A)p(B)p(C = c|A, B) \sum_{D} p(D|B, C = c) \sum_{E} p(E|C = c, D) \\ &= \sum_{B} p(A)p(B)p(C = c|A, B) \qquad \text{[4 terms]} \end{aligned}$$

Total: 4+2+2 = 8 terms

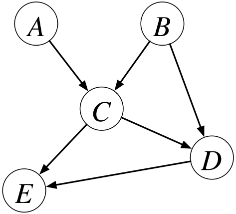
Belief propagation methods use the conditional independence relationships in a graph to do efficient inference (for singly connected graphs, exponential gains in efficiency!).

Belief Propagation (in singly connected DAGs)

Definition: A DAG is *singly connected* if its underlying undirected graph is a tree, *ie* there is only one undirected path between any two nodes.



singly connected

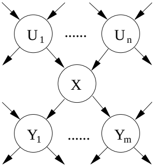


multiply connected

Goal: For some node X we want to compute $p(X|e)$ given evidence (i.e. observed, visible variables) e.
Since we are considering singly connected graphs:

- every node X divides the evidence into upstream e^+_X and downstream e^-_X
- every edge $X \to Y$ divides the evidence into upstream e^+_{XY} and downstream e^-_{XY}.

Three key ideas behind Belief Propagation



Idea 1: The probability of a variable X can be found by combining upstream and downstream evidence:

$$p(X|e) = \frac{p(X,e)}{p(e)} = \frac{p(X,e_X^+,e_X^-)}{p(e_X^+,e_X^-)} \propto p(X|e_X^+) \times \underbrace{p(e_X^-|X,e_X^+)}_{X \text{ d-separates } e_X^- \text{ from } e_X^+}$$

$$= p(X|e_X^+)p(e_X^-|X) = \pi(X)\lambda(X)$$

Idea 2: The upstream and downstream evidence can be computed via a local message passing algorithm between the nodes in the graph.
Idea 3: "Don't send back to a node (any part of) the message it sent to you!"

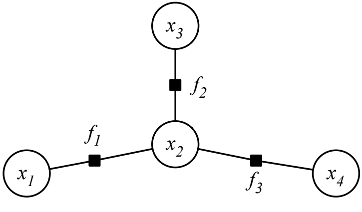
we will focus on factor graphs (simpler) instead of DAGs...

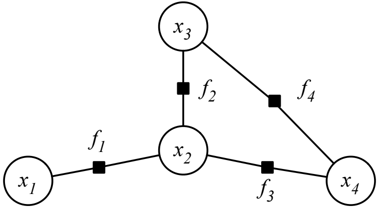
Factor Graphs

Algorithmically and implementationally, it's often easier to convert directed and undirected graphs into factor graphs, and run *factor graph propagation*.

$$
\begin{aligned}
p(\mathbf{x}) &= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) \\
&\equiv f_1(x_1, x_2)f_2(x_2, x_3)f_3(x_2, x_4)
\end{aligned}
$$

Singly connected vs Multiply connected factor graphs:



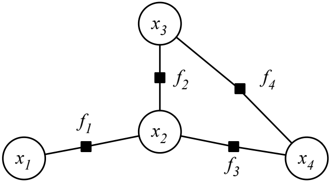


Factor graphs

In a factor graph, the joint probability distribution is written as a product of factors. Consider a vector of variables $\mathbf{x} = (x_1, \ldots, x_n)$

$$p(\mathbf{x}) = p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_j f_j(\mathbf{x}_{S_j})$$

where Z is the normalisation constant, S_j denotes the subset of $\{1, \ldots, n\}$ which participate in factor f_j and $\mathbf{x}_{S_j} = \{x_i : i \in S_j\}$.



variables nodes: we draw open circles for each variable x_i in the distribution.
factor nodes: we draw filled dots for each factor f_j in the distribution.

Propagation in Factor Graphs

Let $n(x)$ denote the set of factor nodes that are neighbors of x.
Let $n(f)$ denote the set of variable nodes that are neighbors of f.
We can compute probabilities in a factor graph by propagating messages from variable nodes to factor nodes and viceversa.

message from variable x to factor f:

$$\mu_{x \to f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \to x}(x)$$

message from factor f to variable x:

$$\mu_{f \to x}(x) = \sum_{\mathbf{x} \setminus x} \left(f(\mathbf{x}) \prod_{y \in n(f) \setminus \{x\}} \mu_{y \to f}(y) \right)$$

where $\mathbf{x}$ are the variables that factor f depends on, and $\sum_{\mathbf{x} \setminus x}$ is a sum over all variables neighboring factor f except x.

Propagation in Factor Graphs

$n(x)$ denotes the set of factor nodes that are neighbors of x.
$n(f)$ denotes the set of variable nodes that are neighbors of f.

message from variable x to factor f:

$$\mu_{x \to f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \to x}(x)$$

message from factor f to variable x:

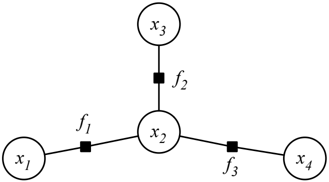
$$\mu_{f \to x}(x) = \sum_{\mathbf{x} \setminus x} \left(f(\mathbf{x}) \prod_{y \in n(f) \setminus \{x\}} \mu_{y \to f}(y) \right)$$

If a variable has only one factor as a neighbor, it can initiate message propagation.

Once a variable has received all messages from its neighboring factor nodes we can compute the probability of that variable by multiplying all the messages and renormalising:

$$p(x) \propto \prod_{h \in n(x)} \mu_{h \to x}(x)$$

Propagation in Factor Graphs



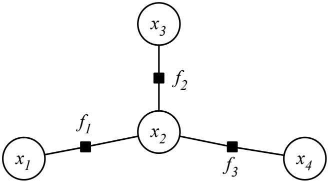
initialise all messages to be constant functions

an example schedule of messages resulting in computing $p(x_4)$:

message direction	message value
$x_1 \rightarrow f_1$	$1(x_1)$
$x_3 \rightarrow f_2$	$1(x_3)$
$f_1 \rightarrow x_2$	$\sum_{x_1} f_1(x_1, x_2) 1(x_1)$
$f_2 \rightarrow x_2$	$\sum_{x_3} f_2(x_3, x_2) 1(x_3)$
$x_2 \rightarrow f_3$	$\left(\sum_{x_1} f_1(x_1, x_2)\right) \left(\sum_{x_3} f_2(x_3, x_2)\right)$
$f_3 \rightarrow x_4$	$\sum_{x_2} f_3(x_2, x_4) \left(\sum_{x_1} f_1(x_1, x_2)\right) \left(\sum_{x_3} f_2(x_3, x_2)\right)$

where $1(x)$ is a constant uniform function of x

Propagation in Factor Graphs

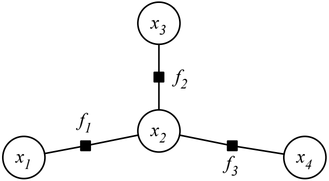


an example schedule of messages resulting in computing $p(x_4|x_1 = a)$:

message direction	message value
$x_1 \to f_1$	$\delta(x_1 = a)$
$x_3 \to f_2$	$1(x_3)$
$f_1 \to x_2$	$\sum_{x_1} f_1(x_1, x_2)\delta(x_1 = a) \; = \; f_1(x_1 = a, x_2)$
$f_2 \to x_2$	$\sum_{x_3} f_2(x_3, x_2)1(x_3)$
$x_2 \to f_3$	$f_1(x_1 = a, x_2) \left(\sum_{x_3} f_2(x_3, x_2)\right)$
$f_3 \to x_4$	$\sum_{x_2} f_3(x_2, x_4) f_1(x_1 = a, x_2) \left(\sum_{x_3} f_2(x_3, x_2)\right)$

where $\delta(x = a)$ is a delta function

Elimination Rules for Factor Graphs



- **eliminating observed variables**
  If a variable x_i is **observed,** i.e. its value is given, then it is a *constant* in all factor that include x_i.

  We can **eliminate** x_i from the graph by removing the corresponding node and modifying all neighboring factors to treat it as a constant.

Elimination Rules for Factor Graphs

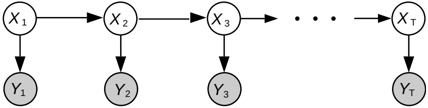
- **eliminating hidden variables**
  If a variable x_i is **hidden** and we are not interested in it we can eliminate it from the graph by summing over all its values.

$$
\begin{aligned}
\sum_{x_i} p(\mathbf{x}) &= \frac{1}{Z} \sum_{x_i} \prod_j f_j(\mathbf{x}_{S_j}) \\
&= \frac{1}{Z} \prod_{j \notin n(x_i)} f_j(\mathbf{x}_{S_j}) \left(\sum_{x_i} \prod_{k \in n(x_i)} f_k(\mathbf{x}_{S_k}) \right) \\
&= \frac{1}{Z} \prod_{j \notin n(x_i)} f_j(\mathbf{x}_{S_j}) \quad f_{\text{new}}(\mathbf{x}_{S_{\text{new}}})
\end{aligned}
$$

where $f_{\text{new}}(\mathbf{x}_{S_{\text{new}}}) = \sum_{x_i} \prod_{k \in n(x_i)} f_k(\mathbf{x}_{S_k})$ and $S_{\text{new}} = \bigcup_{k \in n(x_i)} S_k \setminus \{i\}$.

This causes all its neighboring factor nodes to merge into one new factor node.

Inference in hidden Markov models (HMMs) and linear Gaussian state-space models (SSMs)



$$p(X_{1,\dots,T}, Y_{1,\dots,T}) = p(X_1)p(Y_1|X_1)\prod_{t=2}^{T}[p(X_t|X_{t-1})p(Y_t|X_t)]$$

- In HMMs, the states X_t are discrete.
- In linear Gaussian SSMs, the states are real Gaussian vectors.
- Both HMMs and SSMs can be represented as singly connected DAGs.
- The forward–backward algorithm in hidden Markov models (HMMs), and the Kalman smoothing algorithm in SSMs are both instances of belief propagation / factor graph propagation.

Inference in multiply connected DAGs

The Junction Tree algorithm: Form an undirected graph from your directed graph such that no additional conditional independence relationships have been created (this step is called "moralization"). Lump variables in cliques together and form a tree of cliques—this may require a nasty step called "triangulation". Do inference in this tree of cliques.

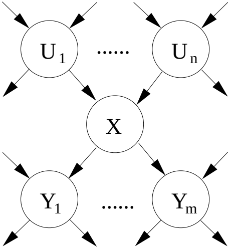
Cutset Conditioning: or "reasoning by assumptions". Find a small set of variables which, if they were given (i.e. known) would render the remaining graph singly connected. For each value of these variables run belief propagation on the singly connected network. Average the resulting beliefs with the appropriate weights (given by normalizing constants).

Loopy Belief Propagation: just use BP although there are loops. In this case the terms "upstream" and "downstream" are not clearly defined. No guarantee of convergence, except for certain special graphs, but often works well in practice (c.f. "turbo-decoding" for error-correcting codes).

Summary

- inference consists of the problem of computing p(variables of interest|observed variables)
- for singly connected DAGs, belief propagation solves this problem exactly.
- for factor graphs, the analogous algorithm is factor graph propagation.
- well-known algorithms such as Kalman smoothing and forward-backward are special cases these general propagation algorithms.
- for multiply connected graphs, the junction tree algorithm solves the exact inference problem, but can be *very* slow (exponential in the cardinality of the largest clique).
- one approximate inference algorithm is "loopy belief propagation"—we will see other approximate inference algorithms in a later lecture.

Appendix: Belief Propagation in Directed Graphs



top-down upstream evidence:
(message U_i sends to X)

$$\pi_X(U_i) = p(U_i | e^+_{U_i X})$$

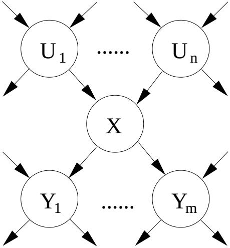
bottom-up downstream evidence:
(message Y_j sends to X)

$$\lambda_{Y_j}(X) = p(e^-_{X Y_j} | X)$$

To update the probability of X given the observed data:

$$\begin{aligned} \mathrm{BEL}(X) = p(X|e) &= \frac{1}{Z}\lambda(X)\,\pi(X) \\ \lambda(X) &= \prod_j \lambda_{Y_j}(X) \\ \pi(X) &= \sum_{U_1 \cdots U_n} p(X|U_1, \ldots, U_n) \prod_i \pi_X(U_i) \end{aligned}$$

Belief Propagation (cont.)



top-down upstream evidence: (message U_i sends to X)

$\pi_X(U_i) = p(U_i|e^+_{U_i X})$

bottom-up downstream evidence: (message Y_j sends to X)

$\lambda_{Y_j}(X) = p(e^-_{XY_j}|X)$

Bottom-up propagation, message X sends to U_i:

$$\lambda_X(U_i) \;=\; \sum_X \lambda(X) \sum_{U_k: k\neq i} p(X|U_1, \ldots, U_n) \prod_{k\neq i} \pi_X(U_k)$$

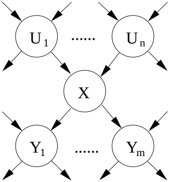
Top-down propagation, message X sends to Y_j:

$$\pi_{Y_j}(X) \;=\; \frac{1}{Z}\Big[\prod_{k\neq j} \lambda_{Y_k}(X)\Big] \sum_{U_1 \cdots U_n} p(X|U_1, \ldots, U_n) \prod_i \pi_X(U_i) \;=\; \frac{1}{Z}\frac{\text{BEL}(X)}{\lambda_{Y_j}(X)}$$

Z is the normaliser ensuring $\sum_X \pi_{Y_j}(X) = 1$

Demo?

Appendix: Understanding BP equations

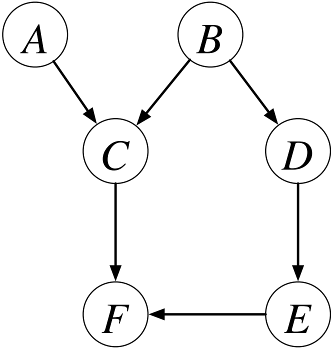


$$
\begin{aligned}
p(X|e) = \text{BEL}(X) &= \tfrac{1}{Z}\lambda(X)\pi(X) = p(e_X^-|X)p(X|e_X^+) && (1)\\
p(e_X^-|X) = \lambda(X) &= \textstyle\prod_j \lambda_{Y_j}(X) = \prod_j p(e_{XY_j}^-|X) && (2)\\
p(X|e_X^+) = \pi(X) &= \textstyle\sum_{U_1 \cdots U_n} p(X|U_1, \ldots, U_n) \prod_i \pi_X(U_i) && (3)\\
&= \textstyle\sum_{U_1 \cdots U_n} p(X|U_1, \ldots, U_n) \prod_i p(U_i|e_{U_i X}^+) && (4)
\end{aligned}
$$

Z is a normalization constant.
All equations follow from the conditional independencies in the graph.

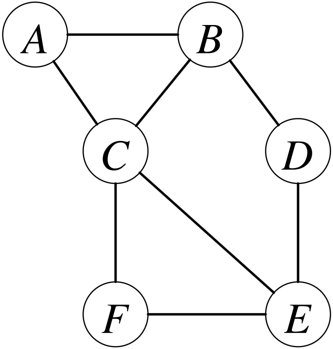
Appendix: The Junction Tree Algorithm

The Junction Tree Algorithm 1



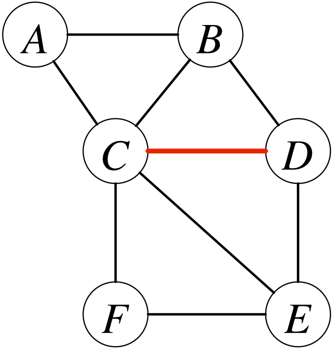
starting with a DAG...

The Junction Tree Algorithm 2



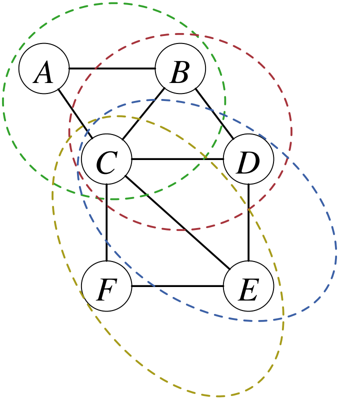
moralize by marrying the parents of each node
remove edge directions
this results in an undirected graph with no additional C.I. relations

The Junction Tree Algorithm 3



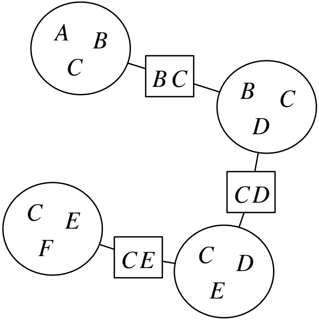
triangulate so that there is no loop of length > 3 without a chord
this is necessary so that the final junction tree satisfies the running intersection property

The Junction Tree Algorithm 4



find cliques of the moralized, triangulated graph

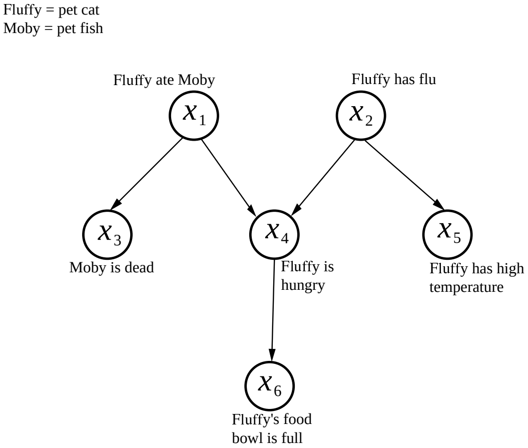
The Junction Tree Algorithm 5



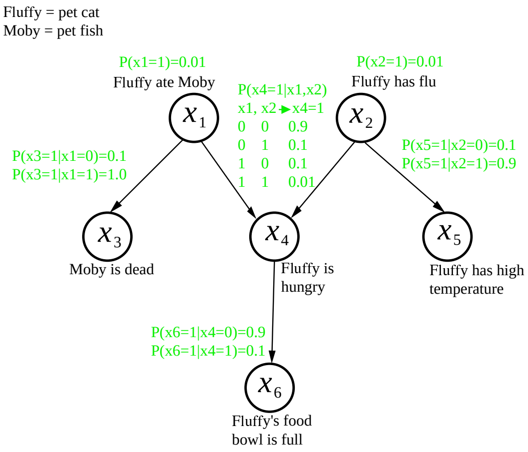
- form **junction tree**: tree of (overlapping) sets of variables
- the **running intersection property** means that if a variable appears in more than one clique (e.g. C), it appears in all intermediate cliques in the tree.
- the junction tree propagation algorithm ensures that neighboring cliques have consistent probability distribution
- local consistency → global consistency

Fluffy and Moby: A Belief Propagation Demo

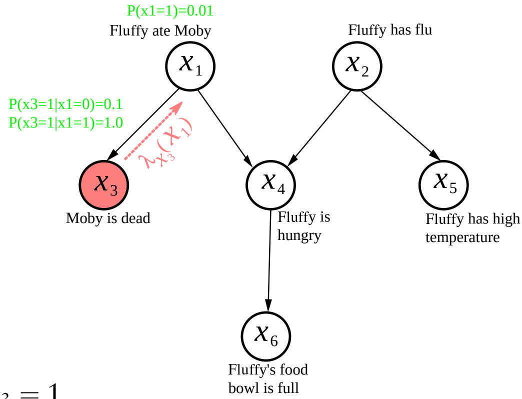
1. Model Structure



2. Model Parameters

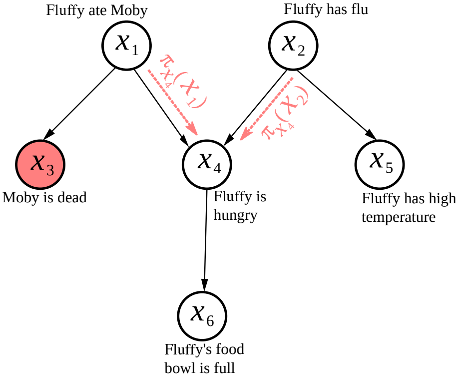


3. Propagating Evidence



1. Observe "Moby is dead", i.e. $x_3 = 1$
2. Send $\lambda_{x_3}(x_1) \equiv p(e^-_{x_1 \to x_3} | x_1) = \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix}$ message $x_3 \to x_1$
3. $\text{BEL}(x_1 | x_3 = 1) = \frac{1}{Z} \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix} \odot \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 0.91 \\ 0.09 \end{bmatrix}$

4. Propagating Evidence



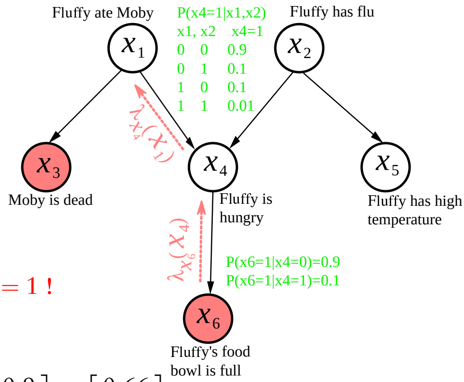
4. Send $\pi_{x_4}(x_1) \equiv p(x_1|e^+_{x_1 \to x_4}) = \begin{bmatrix} 0.91 \\ 0.09 \end{bmatrix}$

5. Send $\pi_{x_4}(x_2) \equiv p(x_2|e^+_{x_2 \to x_4}) = p(x_2) = \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix}$ from $x_2 \to x_4$.

6. Compute $\pi(x_4) \equiv p(x_4|e^+_{x_4}) = \sum_{x_1,x_2} p(x_4|x_1,x_2)\pi_{x_4}(x_1)\pi_{x_4}(x_2) = \begin{bmatrix} 0.18 \\ 0.82 \end{bmatrix}$

7. $\mathrm{BEL}(x_4|x_3 = 1) = \begin{bmatrix} 0.18 \\ 0.82 \end{bmatrix}$, whereas before observing $x_3 = 1$, $\mathrm{BEL}(x_4) = \begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}$.

5. Propagating Evidence



8. Observe "Fluffy's Food Bowl is Full" $x_6 = 1$!
9. Send $\lambda_{x_6}(x_4) = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$ message $x_6 \to x_4$
10. $\text{BEL}(x_4|x_3 = 1, x_6 = 1) = \frac{1}{Z} \begin{bmatrix} 0.18 \\ 0.82 \end{bmatrix} \odot \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.66 \\ 0.34 \end{bmatrix}$
11. Send $\lambda_{x_4}(x_1) = \sum_{x_4} \lambda_{x_6}(x_4) \sum_{x_2} p(x_4|x_1, x_2)\pi_{x_4}(x_2) = \begin{bmatrix} 0.19 \\ 0.82 \end{bmatrix}$
12. $\text{BEL}(x_1|x_3 = 1, x_6 = 1) = \frac{1}{Z} \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix} \odot \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix} \odot \begin{bmatrix} 0.19 \\ 0.82 \end{bmatrix} = \begin{bmatrix} 0.70 \\ 0.30 \end{bmatrix} \Rightarrow$ Fluffy still innocent!