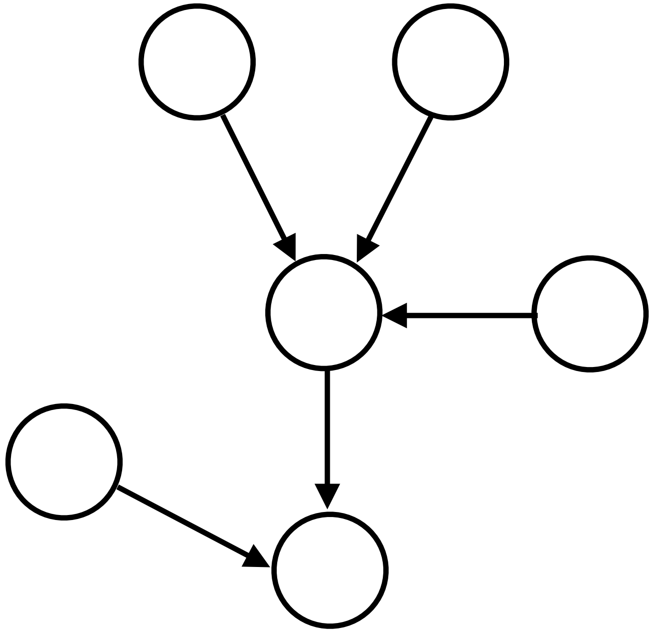
CENTRE FOR ARTIFICIAL INTELLIGENCE

Message Passing Algorithms in Machine Learning

So Takao
so.takao@ucl.ac.uk
www.sotakao.com

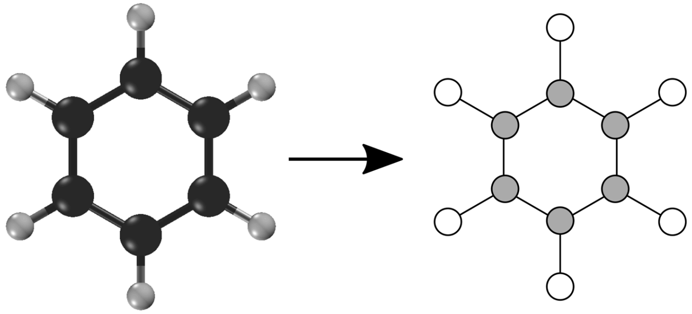
What we will cover in this lecture

We will study machine learning algorithms on **graphs**



Belief network

Images

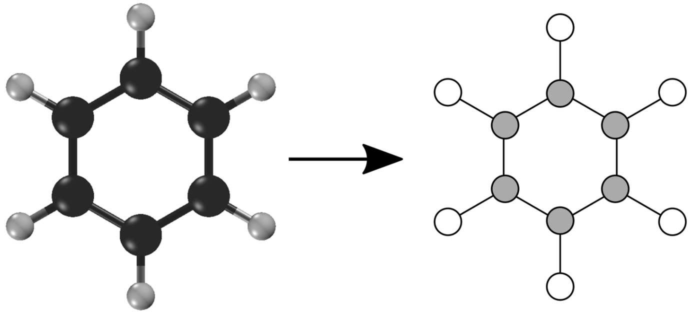


Molecules

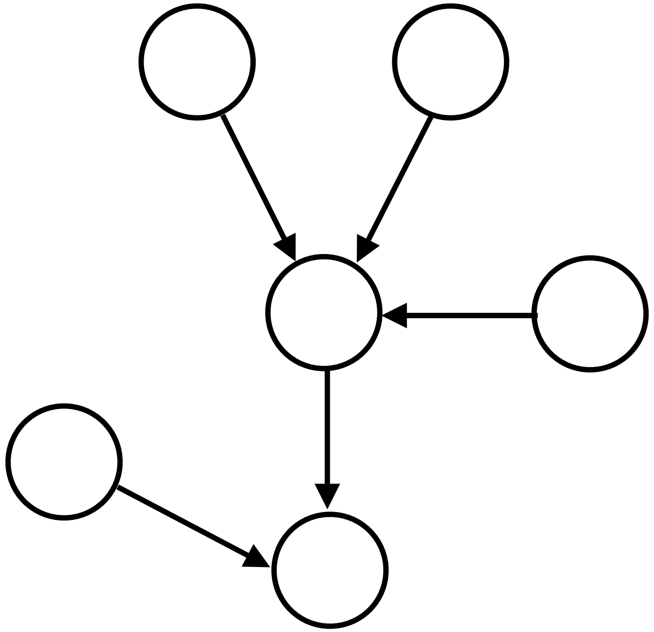
Social networks

What we will cover in this lecture

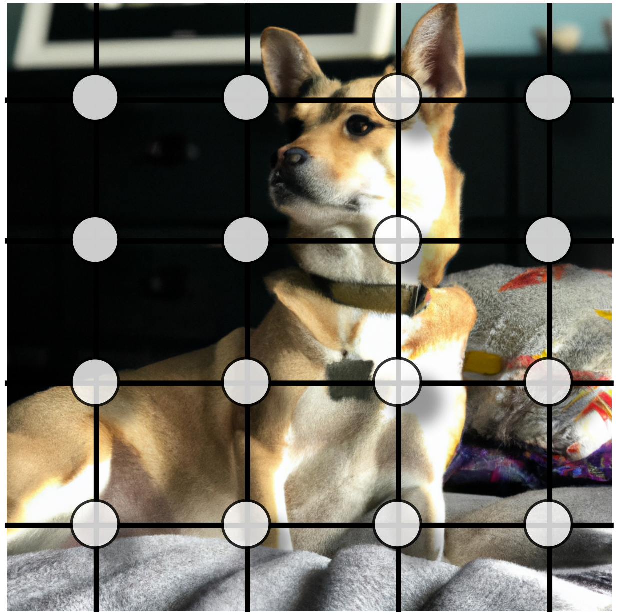
We will study machine learning algorithms on **graphs**



Molecules



Belief network



Images

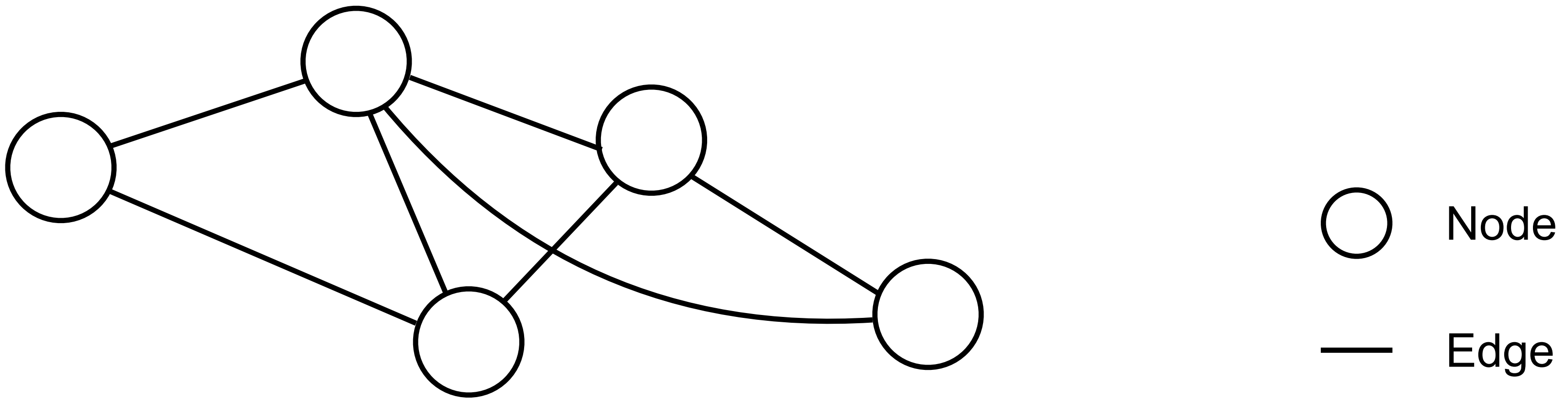
Social networks

What are graphs?

A **Graph** is a collection (V, E) of

- V: nodes
- E: edges

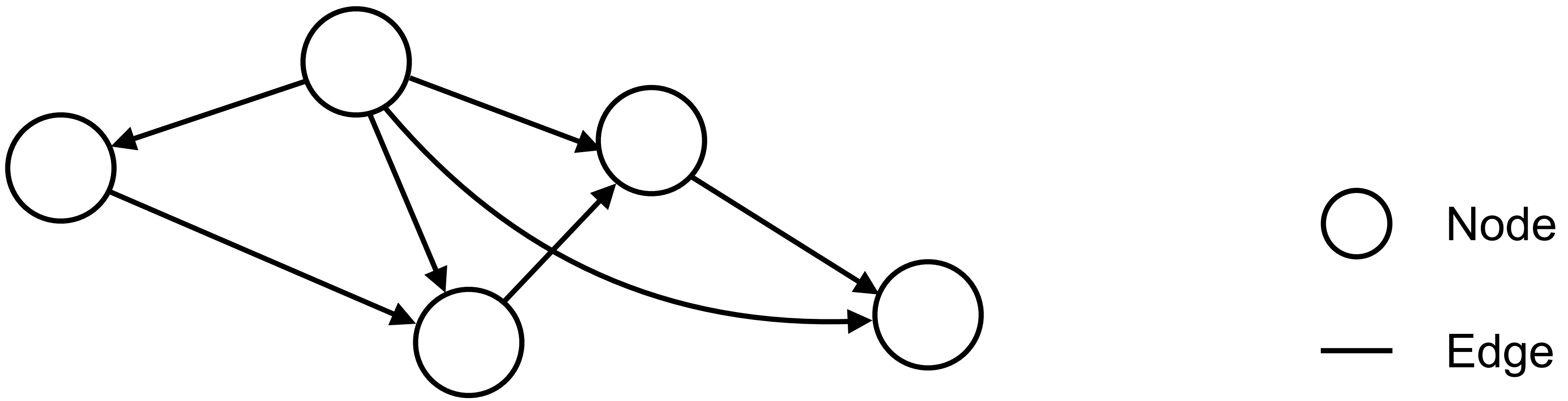
such that an edge $e \in E$ can be associated with a pair of nodes $u, v \in V$.



A graph

- A graph is *directed* if the ordering of nodes associated to an edge "matters"

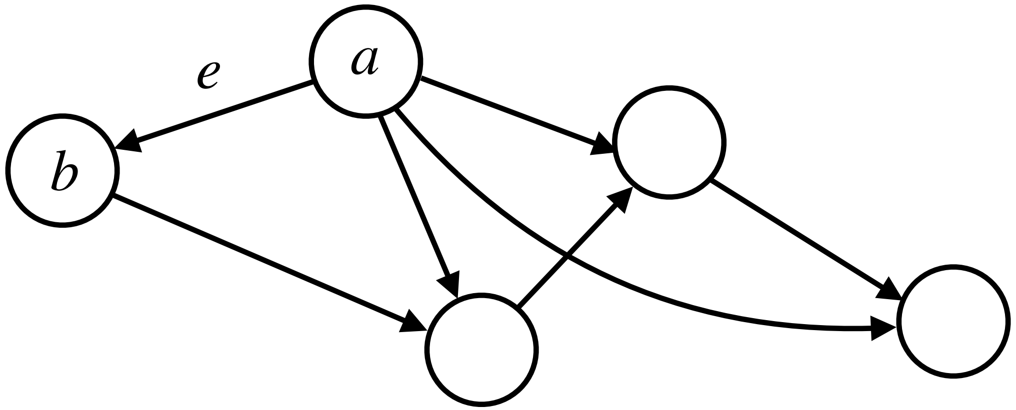
i.e., $\exists\, \phi : E \rightarrow V \times V$ mapping an edge to an *ordered tuple* of nodes.



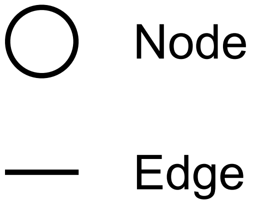
A directed graph

- A graph is *directed* if the ordering of nodes associated to an edge "matters"

i.e., $\exists\, \phi : E \to V \times V$ mapping an edge to an *ordered tuple* of nodes.



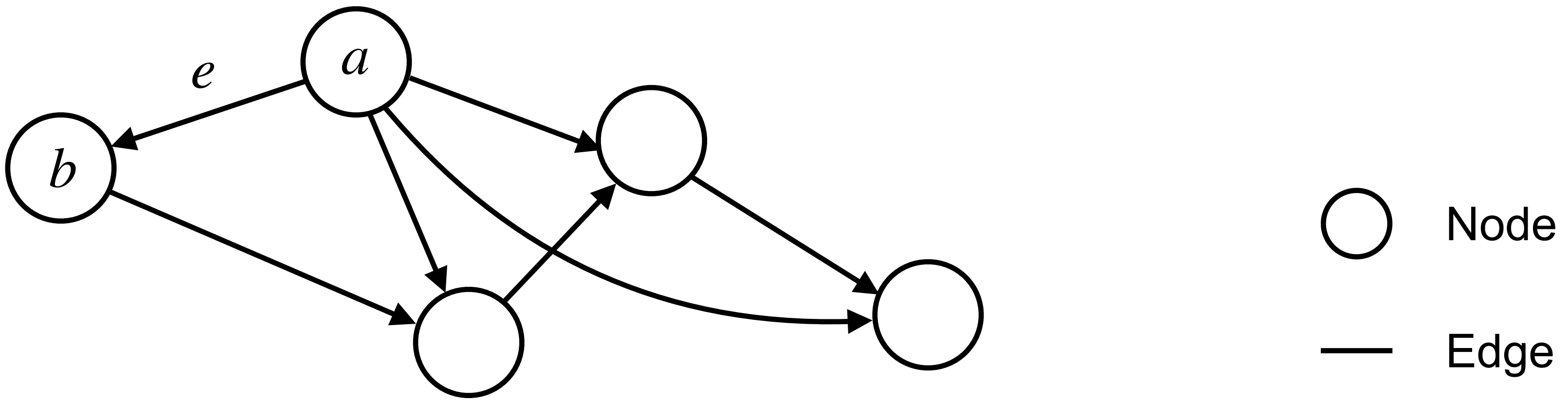
A directed graph



- Edges $\phi(e) = (a, b)$ in a directed graph represented graphically as arrows

- A graph is *directed* if the ordering of nodes associated to an edge “matters”

i.e., $\exists\, \phi : E \rightarrow V \times V$ mapping an edge to an *ordered tuple* of nodes.



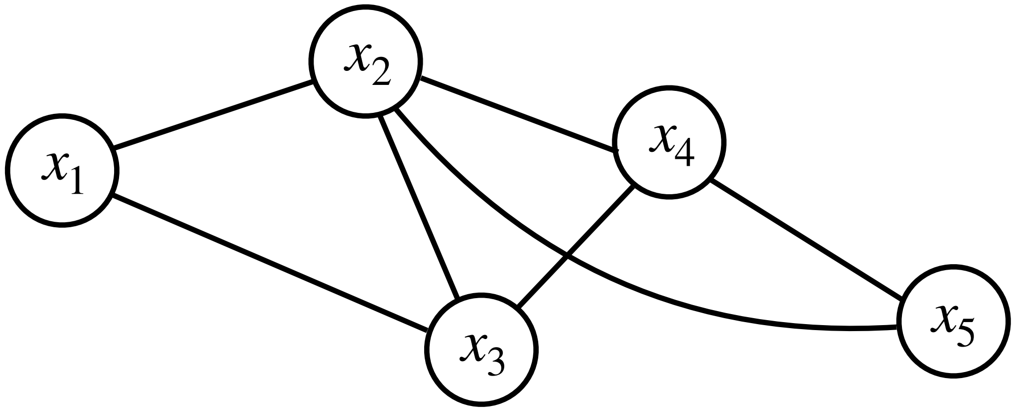
A directed graph

- Edges $\phi(e) = (a, b)$ in a directed graph represented graphically as arrows
- A graph is *undirected* if ordering of nodes in an edge doesn’t matter

- The edges E of a graph define an **adjacency relation** $\sim$ on V:

  For $x, y \in V$,

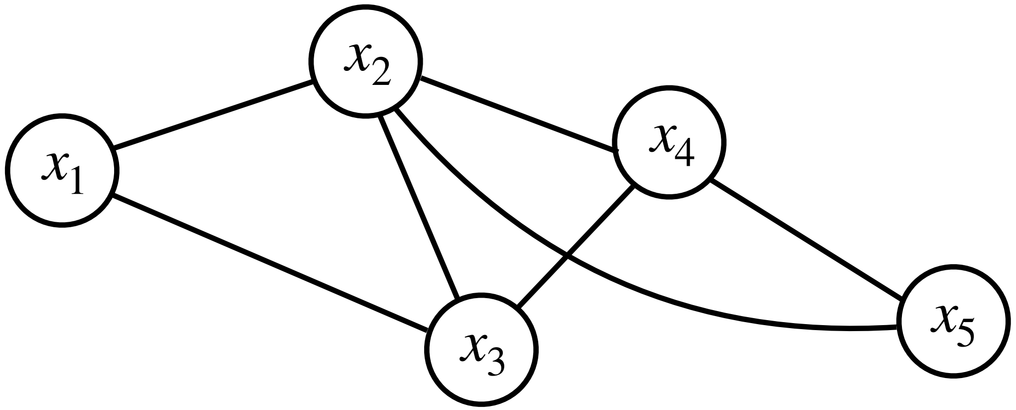
$$x \sim y \quad \Leftrightarrow \quad \{(x, y)\} \cup \{(y, x)\} \subset \phi(E).$$



- The edges E of a graph define an **adjacency relation** $\sim$ on V:

  For $x, y \in V$,

$$x \sim y \quad \Leftrightarrow \quad \{(x,y)\} \cup \{(y,x)\} \subset \phi(E).$$



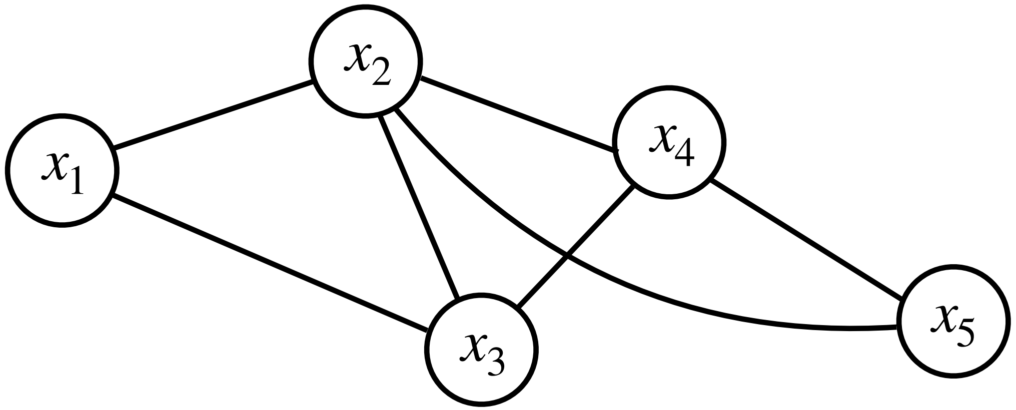
On the graph on the left, we have e.g.

- $x_1 \sim x_2$
- $x_4 \sim x_5$
- $x_1 \nsim x_4$
- $x_3 \nsim x_5$

- The edges E of a graph define an **adjacency relation** $\sim$ on V:

  For $x, y \in V$,

$$x \sim y \quad \Leftrightarrow \quad \{(x,y)\} \cup \{(y,x)\} \subset \phi(E).$$



On the graph on the left, we have e.g.

- $x_1 \sim x_2$
- $x_4 \sim x_5$
- $x_1 \nsim x_4$
- $x_3 \nsim x_5$

- If $x \sim y$, we say that y is a *neighbour* of x and vice versa

- Adjacency matrix $\mathbf{A}$ encodes the adjacency structure of G:

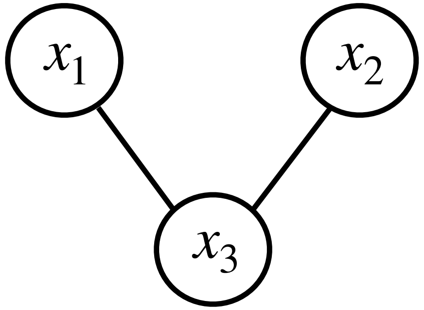
$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if } x_i \sim x_j, \\ 0, & \text{if } x_i \nsim x_j. \end{cases}$$

- Degree matrix $\mathbf{D}$ encodes the degree of connectivity of each node:

$$\mathbf{D}_{ij} = \begin{cases} |\text{Neighbours}(x_i)|, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

- Adjacency matrix $\mathbf{A}$ encodes the adjacency structure of G:

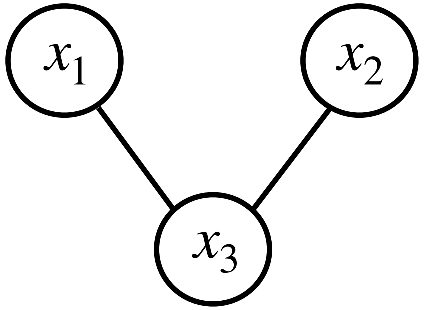
$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if } x_i \sim x_j, \\ 0, & \text{if } x_i \nsim x_j \,. \end{cases}$$



- Degree matrix $\mathbf{D}$ encodes the degree of connectivity of each node:

$$\mathbf{D}_{ij} = \begin{cases} |\,\text{Neighbours}(x_i)\,|\,, & \text{if } i = j, \\ 0, & \text{if } i \neq j \,. \end{cases}$$

- Adjacency matrix $\mathbf{A}$ encodes the adjacency structure of G:

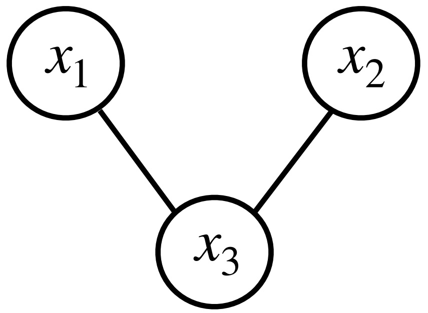


$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

- Degree matrix $\mathbf{D}$ encodes the degree of connectivity of each node:

$$\mathbf{D}_{ij} = \begin{cases} |\,\text{Neighbours}(x_i)\,|\,, & \text{if } i = j, \\ 0, & \text{if } i \neq j\,. \end{cases}$$

- Adjacency matrix $\mathbf{A}$ encodes the adjacency structure of G:

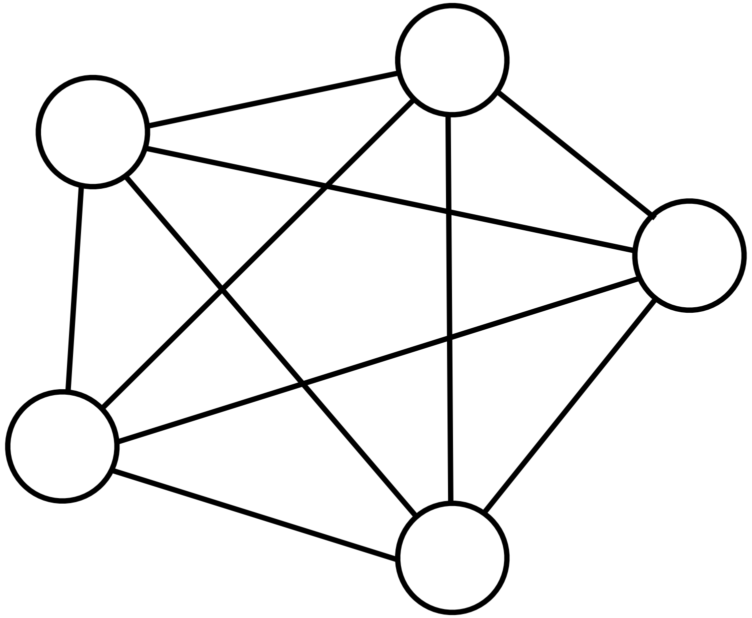


$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

- Degree matrix $\mathbf{D}$ encodes the degree of connectivity of each node:

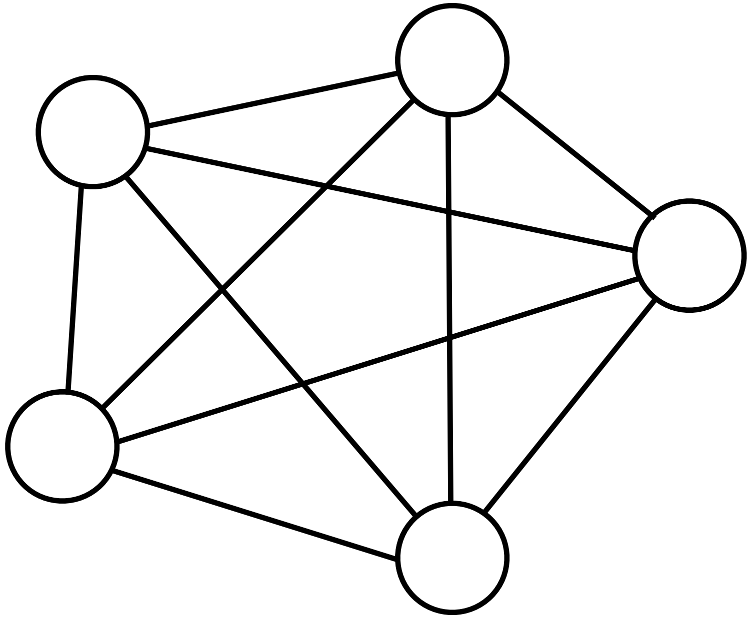
$$\mathbf{D} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Types of Graphs



1. **Fully-connected graphs**

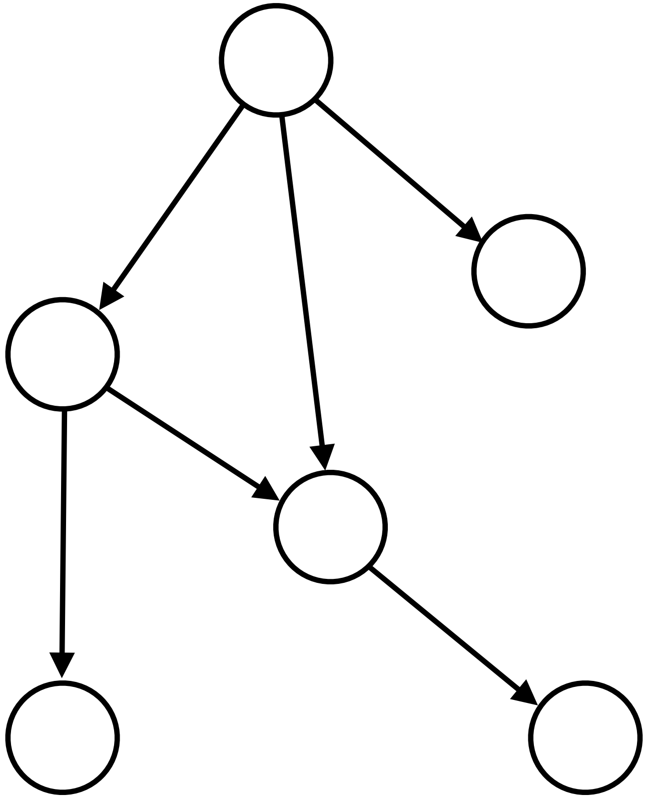
Types of Graphs



1. Fully-connected graphs

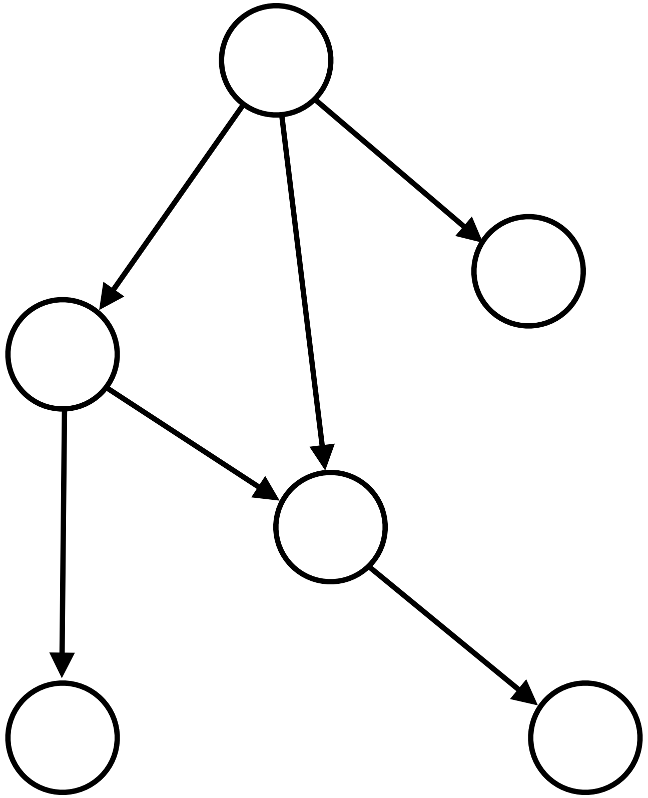
- Undirected
- Each node is connected to every other nodes

Types of Graphs



2. Directed Acyclic Graph (DAG)

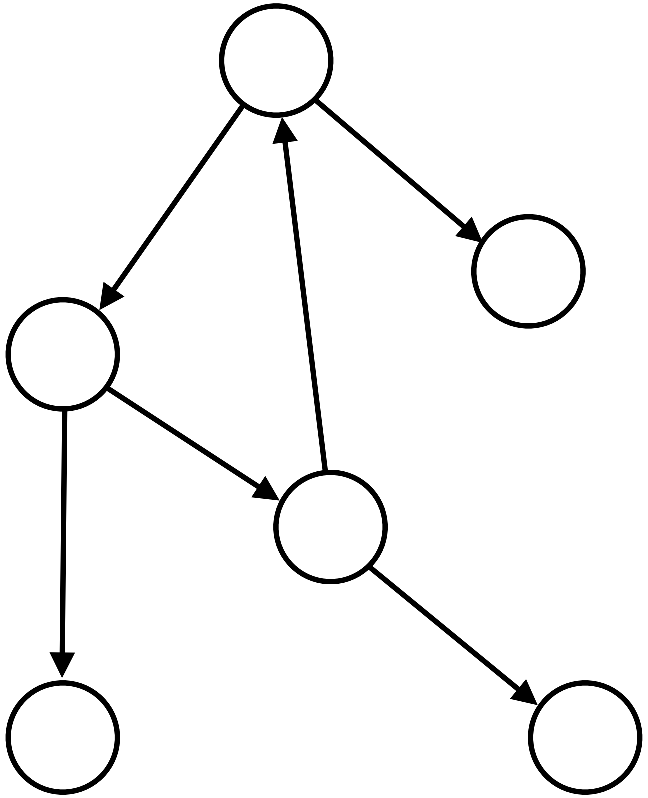
Types of Graphs



2. Directed Acyclic Graph (DAG)

- Directed
- Does not contain any *directed* cycles

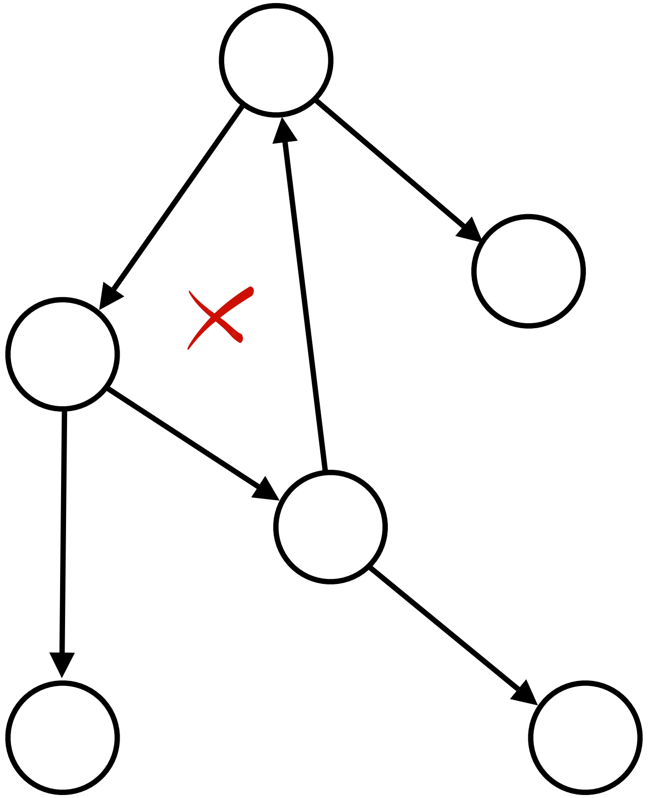
Types of Graphs



2. Directed Acyclic Graph (DAG)

- Directed
- Does not contain any *directed* cycles

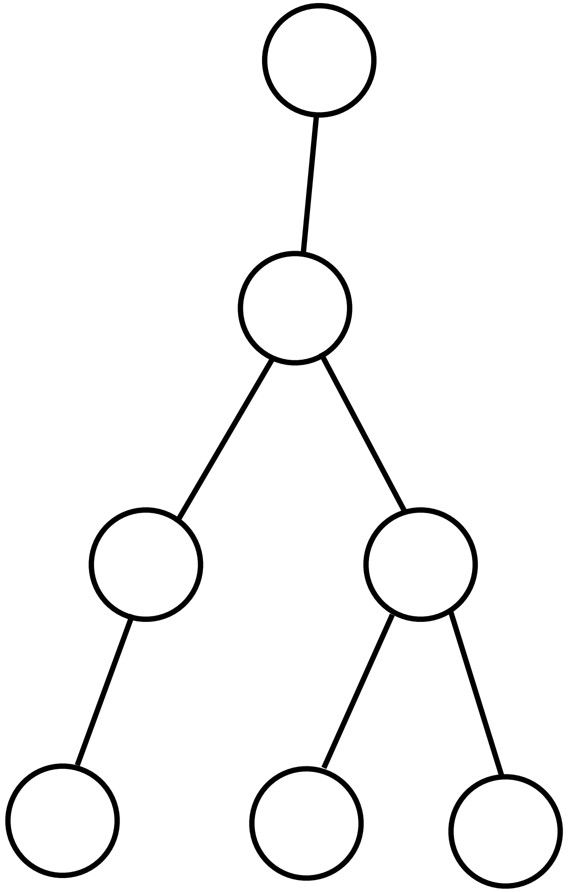
Types of Graphs



2. Directed Acyclic Graph (DAG)

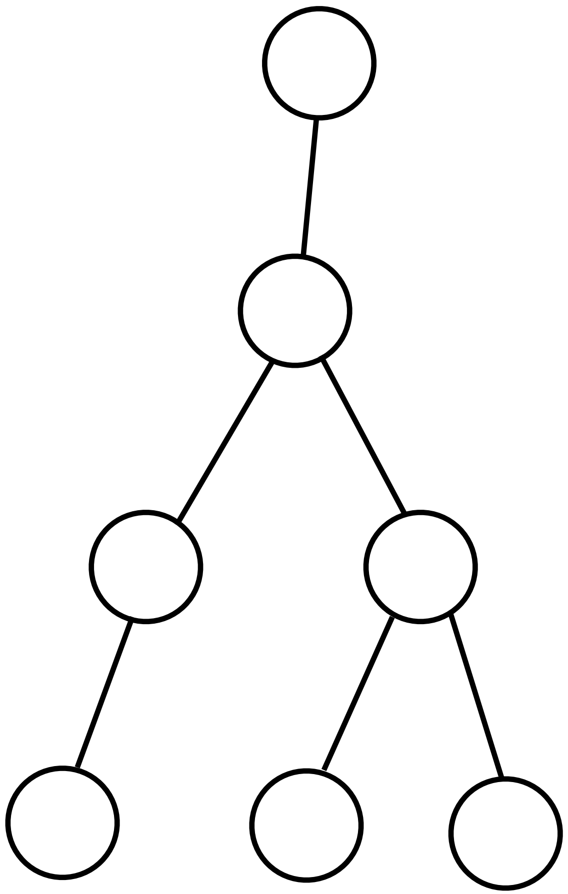
- Directed
- Does not contain any *directed* cycles

Types of Graphs



3. Trees and polytrees

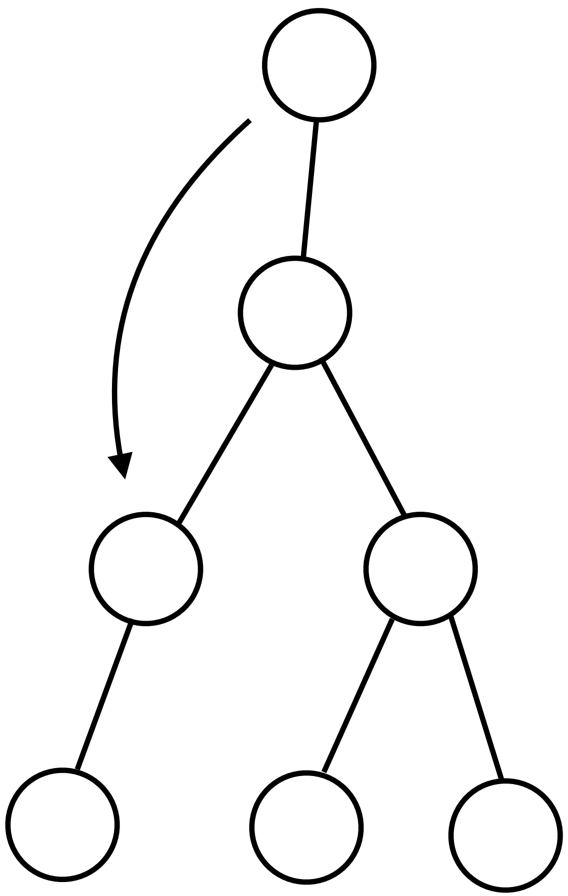
Types of Graphs



3. Trees and polytrees

- A tree is an *undirected* graph such that two nodes are connected by a unique path

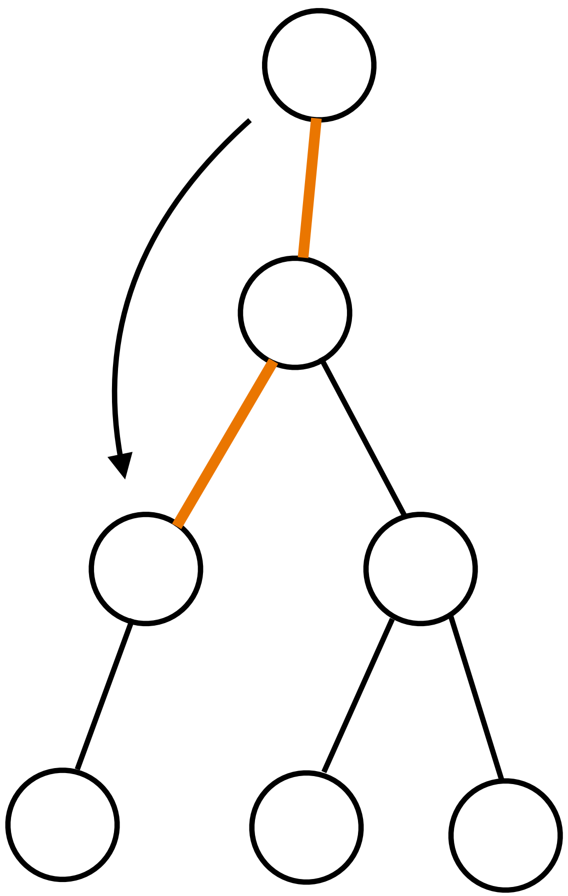
Types of Graphs



3. Trees and polytrees

- A tree is an *undirected* graph such that two nodes are connected by a unique path

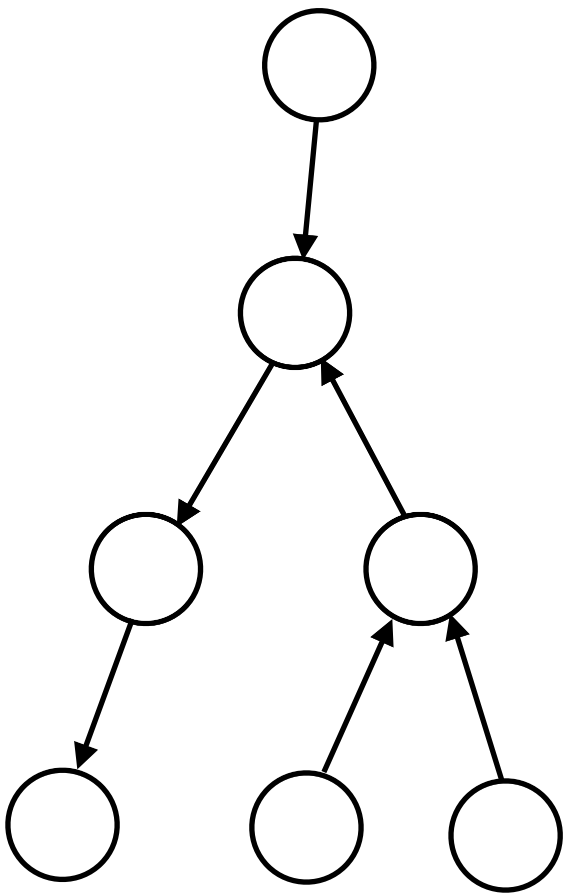
Types of Graphs



3. Trees and polytrees

- A tree is an *undirected* graph such that two nodes are connected by a unique path

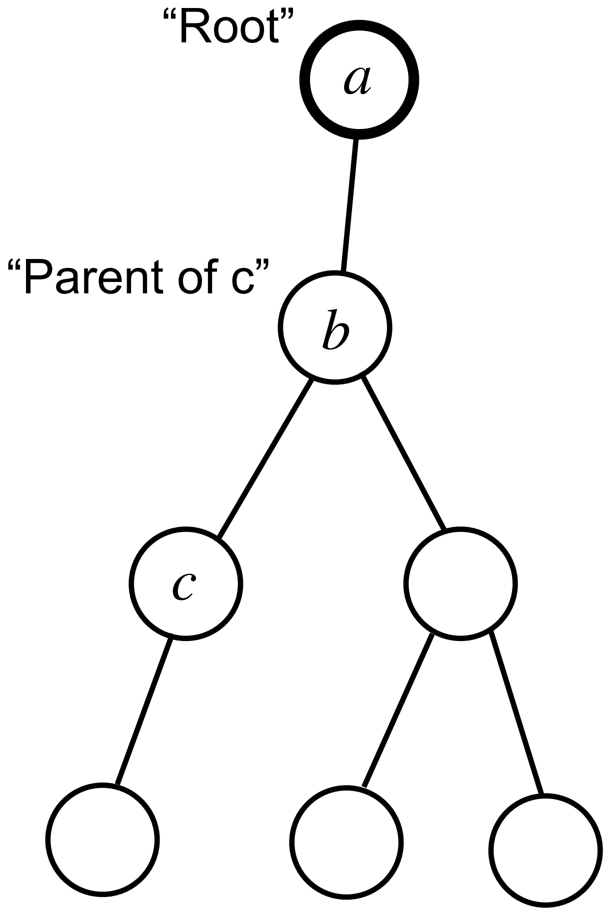
Types of Graphs



3. Trees and polytrees

- A tree is an *undirected* graph such that two nodes are connected by a unique path
- A polytree is a *DAG* such that its underlying structure is a tree

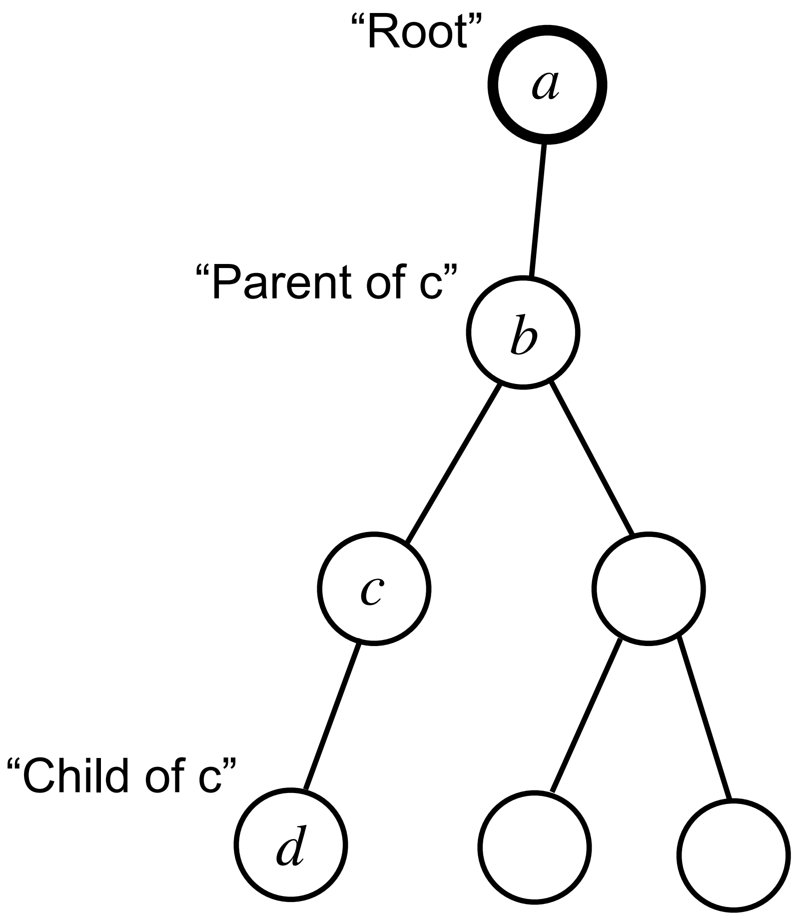
Types of Graphs



3. Trees and polytrees

- A tree is an *undirected* graph such that two nodes are connected by a unique path
- A polytree is a *DAG* such that its underlying structure is a tree
- Designating node a as a “root”, we say that node b is a *parent* of node c if it is a neighbouring node *on the path to* a

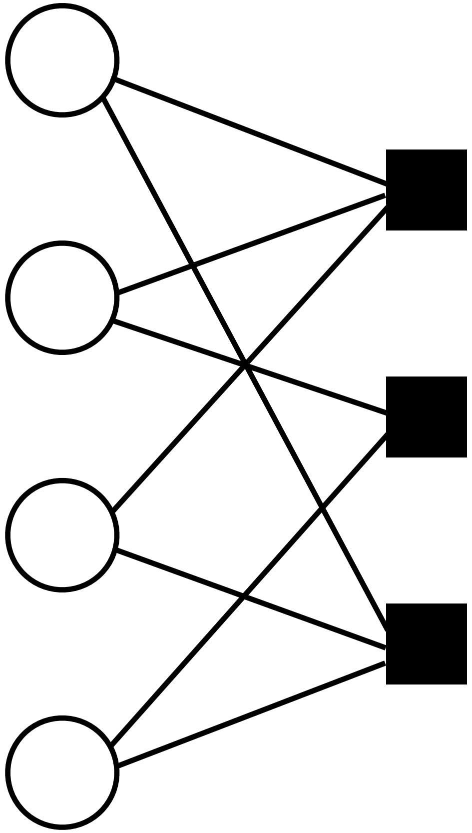
Types of Graphs



3. Trees and polytrees

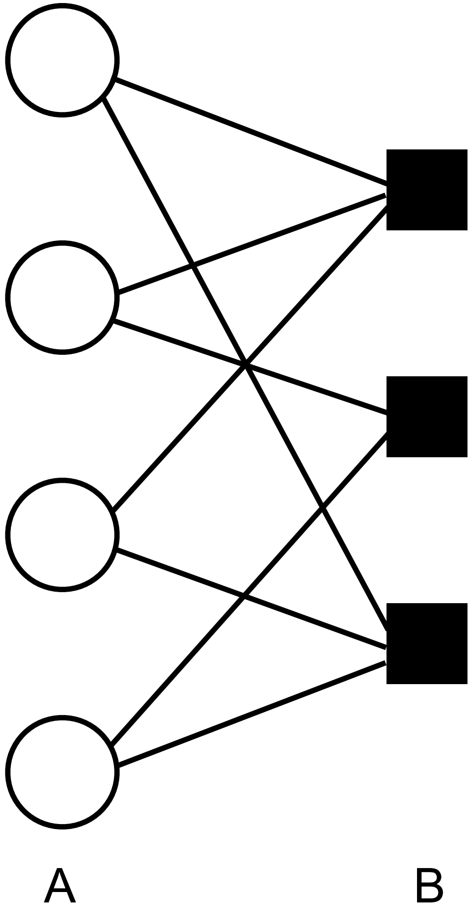
- A tree is an *undirected* graph such that two nodes are connected by a unique path
- A polytree is a *DAG* such that its underlying structure is a tree
- Designating node a as a “root”, we say that node b is a *parent* of node c if it is a neighbouring node *on the path to* a
- Likewise d is a *child* of c if c is it’s parent

Types of Graphs



4. Bipartite graphs

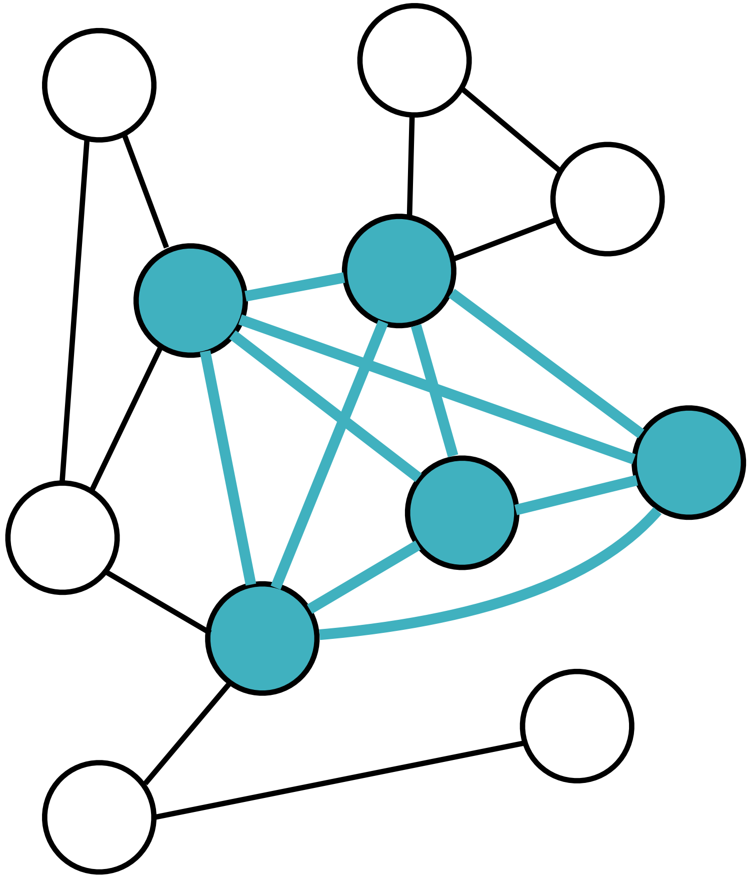
Types of Graphs



4. Bipartite graphs

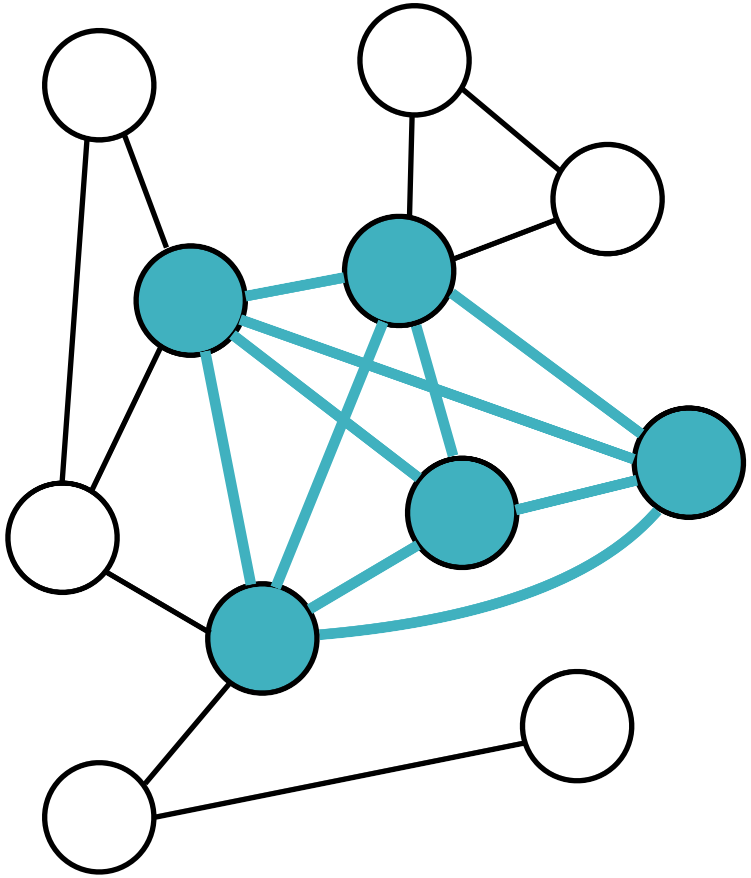
- Nodes can be divided into two “classes” (say A and B)
- Each edge connects a node in A with a node in B
- Can be either directed or undirected

Types of Graphs



5. Subgraphs

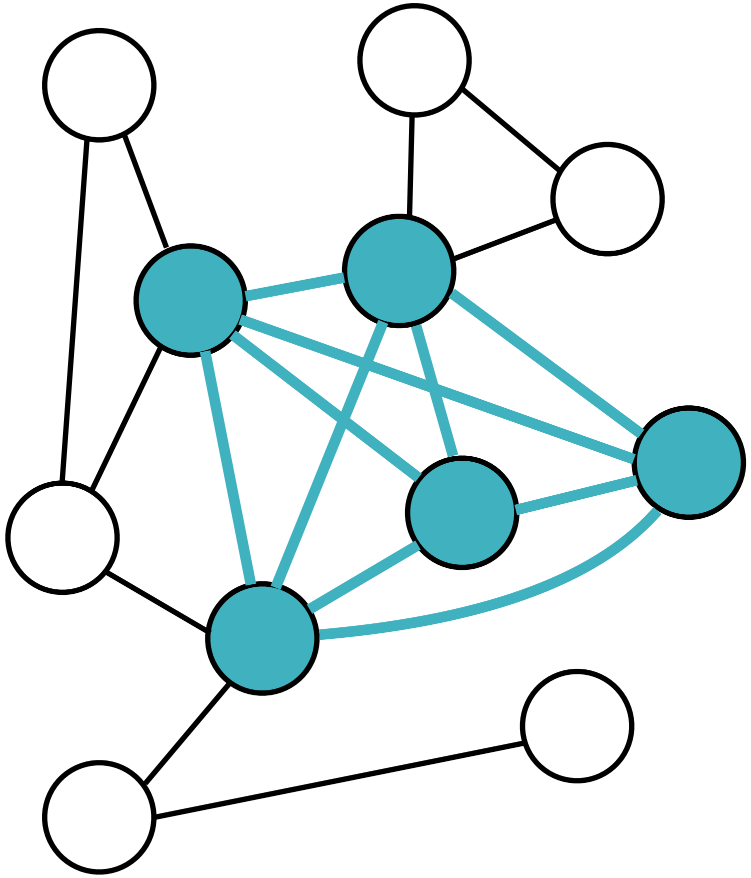
Types of Graphs



5. Subgraphs

Let $G = (V, E)$ be a graph.

Types of Graphs

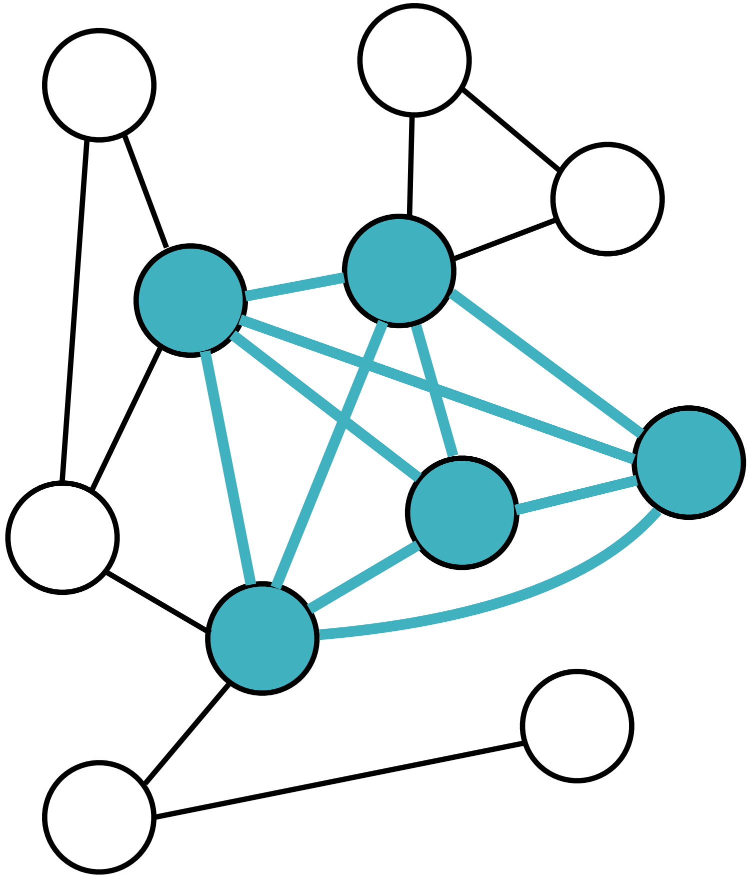


5. Subgraphs

Let $G = (V, E)$ be a graph.

- A subgraph $G_1 = (V_1, E_1)$ of G is a graph such that $V_1 \subset V$ and $E_1 \subset E$

Types of Graphs



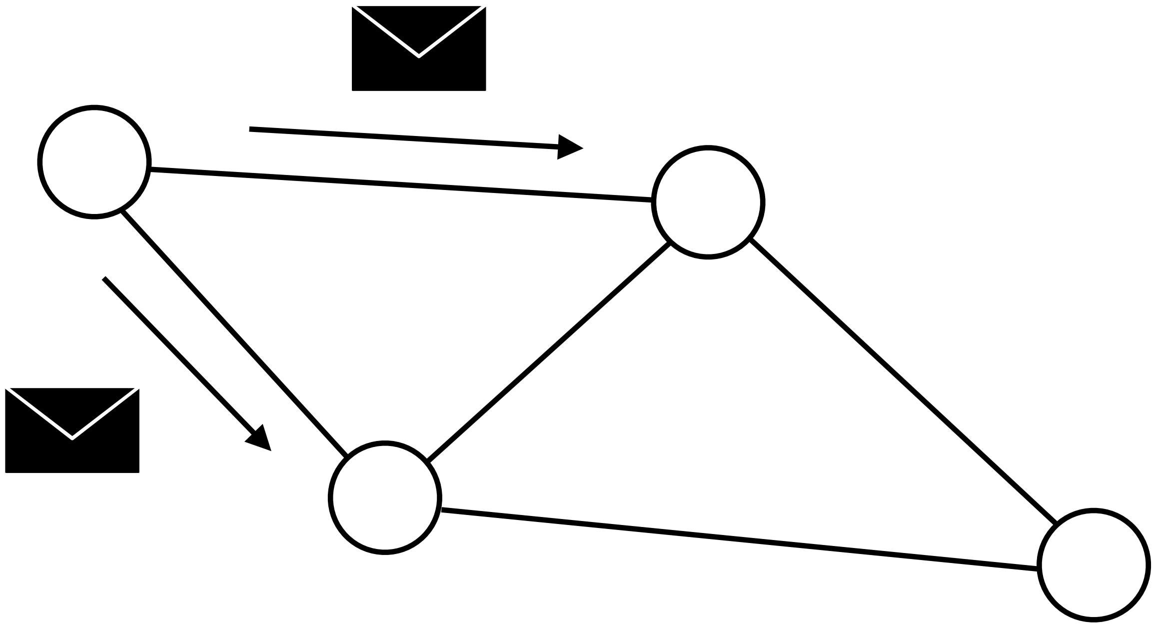
5. Subgraphs

Let $G = (V, E)$ be a graph.

- A subgraph $G_1 = (V_1, E_1)$ of G is a graph such that $V_1 \subset V$ and $E_1 \subset E$
- If a subgraph is *fully-connected*, then we call it a **clique**

Message passing

Algorithms defined on graphs where information is passed between neighbours



Topics covered in this lecture

1. Probabilistic graphical models (PGMs)
2. Belief propagation on PGMs
3. Some extensions of belief propagation
4. Message passing neural networks

Supplementary materials

- Github link: https://github.com/sotakao/ml-seminar-ucl
- References provided at the end of each section
- See Bishop's book [1] for necessary background in graphs and probability theory

[1] Bishop, Christopher M. *Pattern Recognition and Machine Learning*. New York: springer, 2006.

1. Probabilistic Graphical Models (PGMs)

Example

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

Example

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

Questions:

Example

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

Questions:

- If x_4 is observed, are the variables x_2 and x_6 independent?

Example

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

Questions:

- If x_4 is observed, are the variables x_2 and x_6 independent?

  i.e., $$p(x_2, x_6 \mid x_4) \stackrel{?}{=} p(x_2 \mid x_4)\,p(x_6 \mid x_4)$$

Example

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

Questions:

- If x_4 is observed, are the variables x_2 and x_6 independent?
  i.e., $p(x_2, x_6 \,|\, x_4) \stackrel{?}{=} p(x_2 \,|\, x_4)\,p(x_6 \,|\, x_4)$
- Which variable should we observe for x_6 and x_7 to be independent?

Example

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

Questions:

- If x_4 is observed, are the variables x_2 and x_6 independent?

  i.e., $p(x_2, x_6 \,|\, x_4) \stackrel{?}{=} p(x_2 \,|\, x_4)\,p(x_6 \,|\, x_4)$

- Which variable should we observe for x_6 and x_7 to be independent?

  i.e., $p(x_6, x_7 \,|\, ?) = p(x_6 \,|\, ?)\,p(x_7 \,|\, ?)$

Example

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

Questions:

- If x_4 is observed, are the variables x_2 and x_6 independent?

  i.e., $$p(x_2, x_6 \mid x_4) \stackrel{?}{=} p(x_2 \mid x_4)\,p(x_6 \mid x_4)$$

- Which variable should we observe for x_6 and x_7 to be independent?

  i.e., $$p(x_6, x_7 \mid ?) = p(x_6 \mid ?)\,p(x_7 \mid ?)$$

Example

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

Questions:

- If x_4 is observed, are the variables x_2 and x_6 independent?

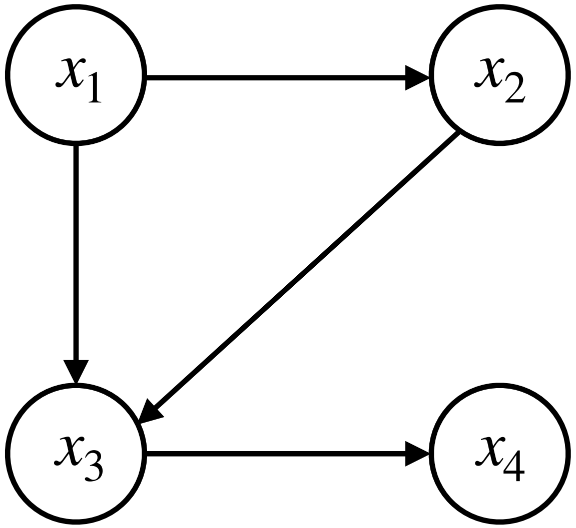
  i.e., $p(x_2, x_6 \,|\, x_4) \stackrel{?}{=} p(x_2 \,|\, x_4)\, p(x_6 \,|\, x_4)$

- Which variable should we observe for x_6 and x_7 to be independent?

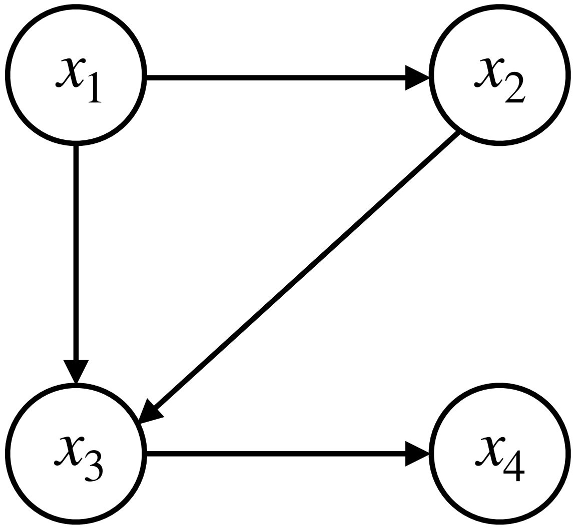
  i.e., $p(x_6, x_7 \,|\, ?) = p(x_6 \,|\, ?)\, p(x_7 \,|\, ?)$

PGMs provide elegant answers to such questions!

Bayesian Networks



Bayesian Networks

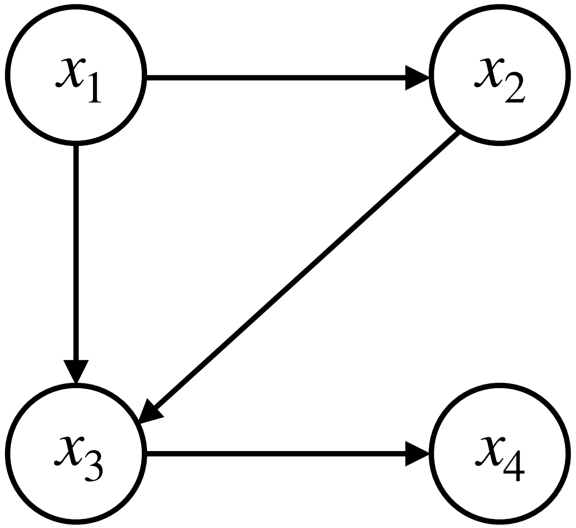


Bayesian networks (BN) visualise how a **joint probability distribution** factorises into **conditional probability distributions**

Example:

$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

Bayesian Networks



Bayesian networks (BN) visualise how a

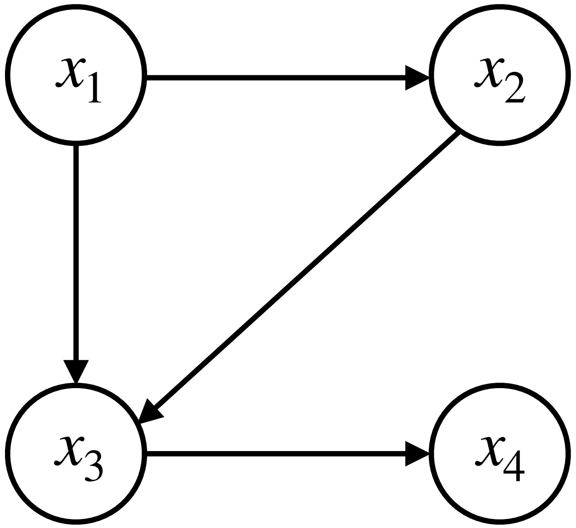
joint probability distribution factorises into **conditional probability distributions**

Example:

$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

- Represented by a **directed acyclic graph (DAG)**

Bayesian Networks



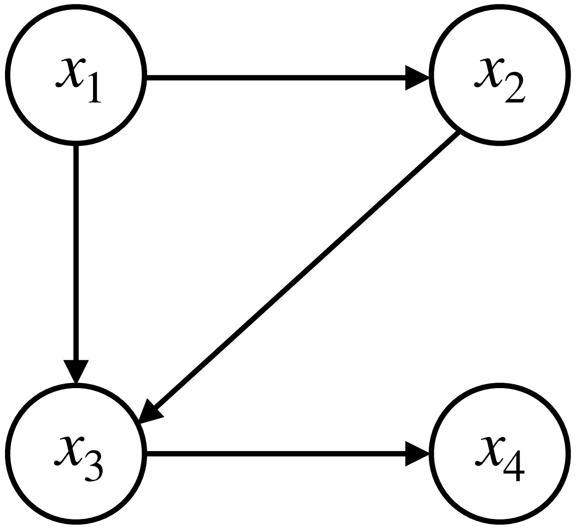
Bayesian networks (BN) visualise how a **joint probability distribution** factorises into **conditional probability distributions**

Example:

$$p(x_1, x_2, x_3, x_4) = p(x_4 \,|\, x_3)\, p(x_3 \,|\, x_1, x_2)\, p(x_2 \,|\, x_1)\, p(x_1)$$

- Represented by a **directed acyclic graph (DAG)**
- Nodes represent variables in the model

Bayesian Networks



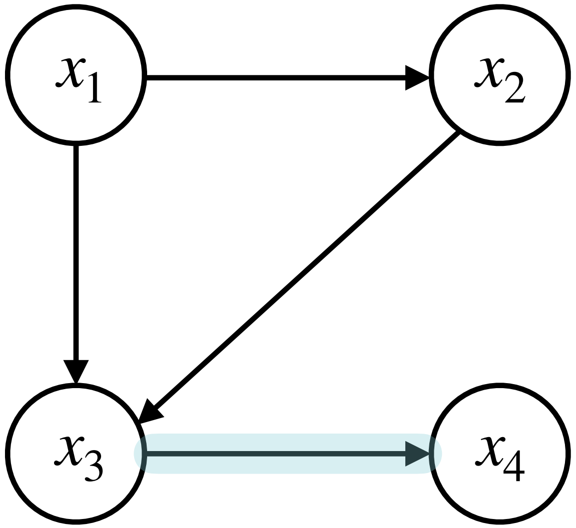
Bayesian networks (BN) visualise how a **joint probability distribution** factorises into **conditional probability distributions**

Example:

$$p(x_1, x_2, x_3, x_4) = p(x_4 \,|\, x_3)\, p(x_3 \,|\, x_1, x_2)\, p(x_2 \,|\, x_1)\, p(x_1)$$

- Represented by a **directed acyclic graph (DAG)**
- Nodes represent variables in the model
- Edges represent causal relations between variables

Bayesian Networks



Bayesian networks (BN) visualise how a

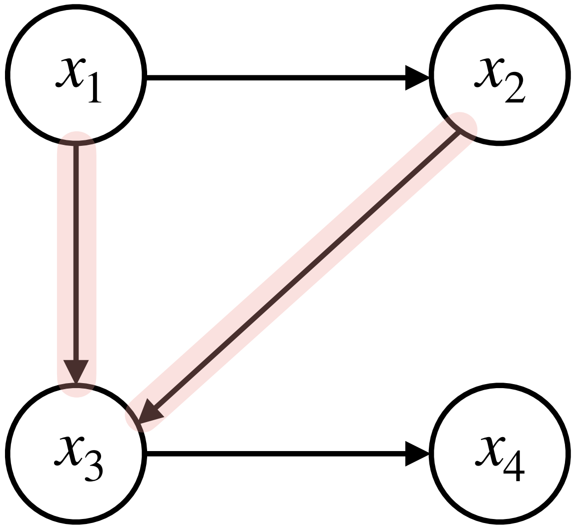
joint probability distribution factorises into **conditional probability distributions**

Example:

$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

- Represented by a **directed acyclic graph (DAG)**
- Nodes represent variables in the model
- Edges represent causal relations between variables

Bayesian Networks



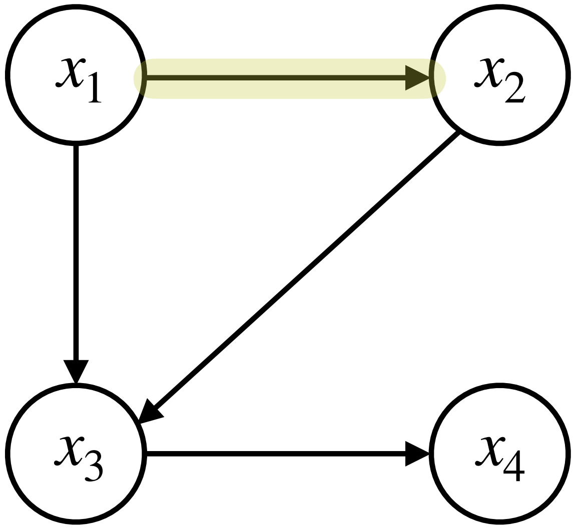
Bayesian networks (BN) visualise how a **joint probability distribution** factorises into **conditional probability distributions**

Example:

$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

- Represented by a **directed acyclic graph (DAG)**
- Nodes represent variables in the model
- Edges represent causal relations between variables

Bayesian Networks



Bayesian networks (BN) visualise how a **joint probability distribution** factorises into **conditional probability distributions**

Example:

$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

- Represented by a **directed acyclic graph (DAG)**
- Nodes represent variables in the model
- Edges represent causal relations between variables

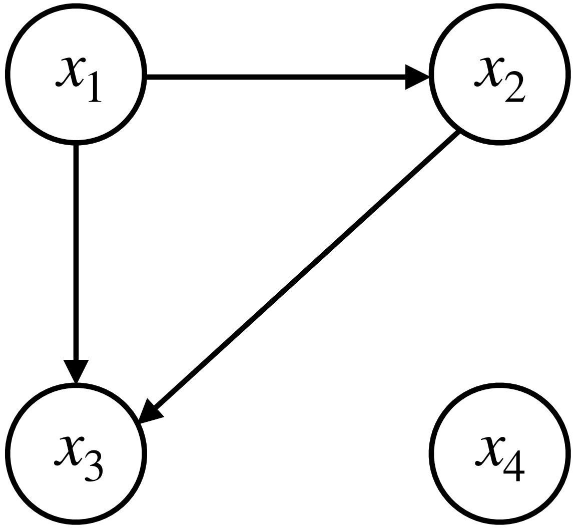
Independence

Independence

Two nodes are *independent* if there are no paths connecting them

Independence

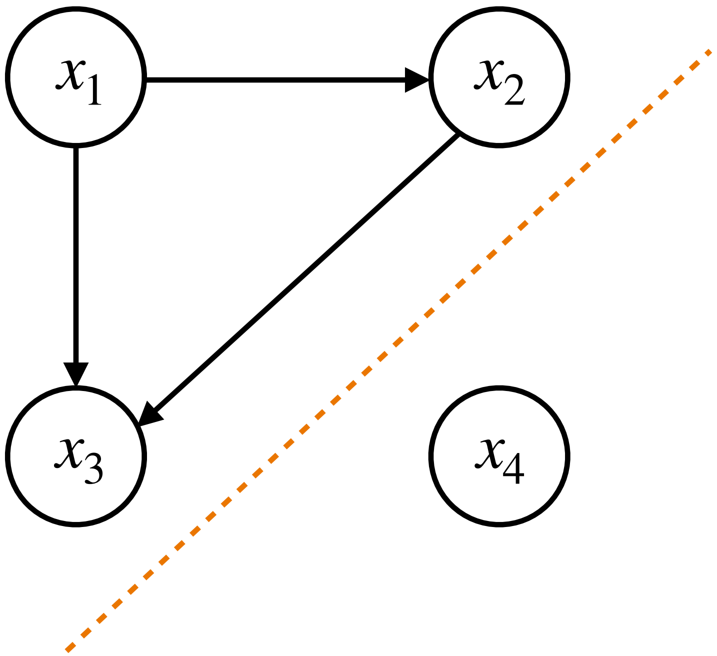
Two nodes are *independent* if there are no paths connecting them



$$
\begin{aligned}
p(x_1, x_2, x_3, x_4) &= p(x_4)\, p(x_3 \,|\, x_1, x_2)\, p(x_2 \,|\, x_1)\, p(x_1) \\
&= p(x_4)\, p(x_1, x_2, x_3)
\end{aligned}
$$

Independence

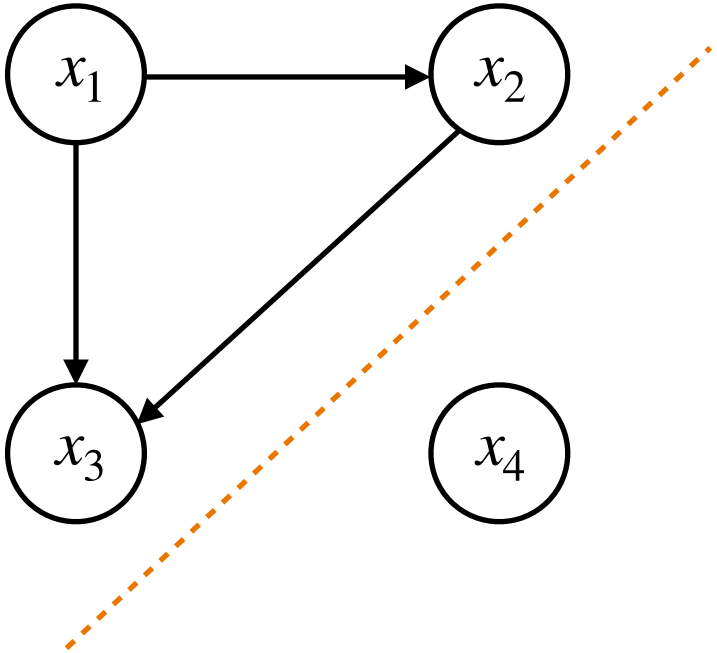
Two nodes are *independent* if there are no paths connecting them



$$\begin{aligned} p(x_1, x_2, x_3, x_4) &= p(x_4)\, p(x_3 \,|\, x_1, x_2)\, p(x_2 \,|\, x_1)\, p(x_1) \\ &= p(x_4)\, p(x_1, x_2, x_3) \end{aligned}$$

Independence

Two nodes are *independent* if there are no paths connecting them



$$p(x_1, x_2, x_3, x_4) = p(x_4)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$
$$= p(x_4)\, p(x_1, x_2, x_3)$$

x_4 is independent of all other nodes

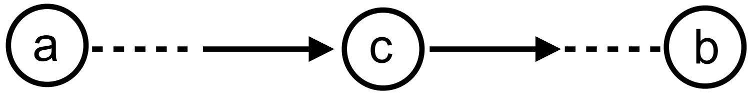
d-Separation and conditional independence

d-Separation and conditional independence

Two nodes a and b in a DAG are **d-separated** by a set of nodes Z if and only if *any* loop-free path from a to b satisfies one of the following:

d-Separation and conditional independence

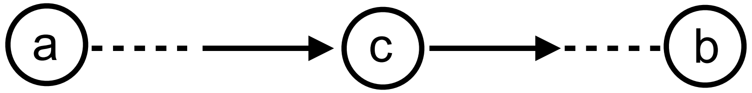
Two nodes a and b in a DAG are **d-separated** by a set of nodes Z if and only if *any* loop-free path from a to b satisfies one of the following:

1. 

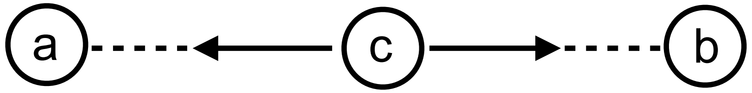
Path contains a **chain** and c belongs to Z.

d-Separation and conditional independence

Two nodes a and b in a DAG are **d-separated** by a set of nodes Z if and only if *any* loop-free path from a to b satisfies one of the following:

1. 

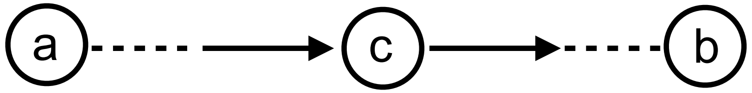
Path contains a **chain** and c belongs to Z.

2. 

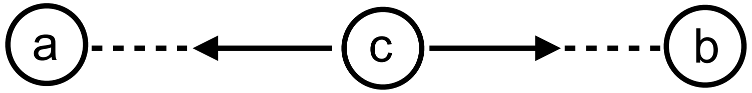
Path contains a **fork** and c belongs to Z.

d-Separation and conditional independence

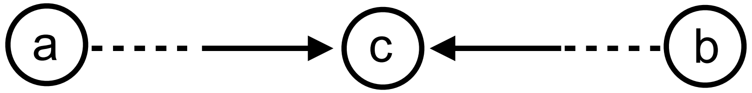
Two nodes a and b in a DAG are **d-separated** by a set of nodes Z if and only if *any* loop-free path from a to b satisfies one of the following:

1. 

Path contains a **chain** and c belongs to Z.

2. 

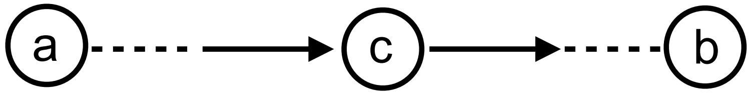
Path contains a **fork** and c belongs to Z.

3. 

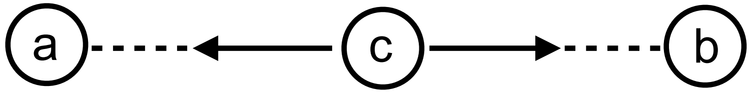
Path contains a **collider** and c *does not* belong to Z. In addition, no descendant of c belongs to Z.

d-Separation and conditional independence

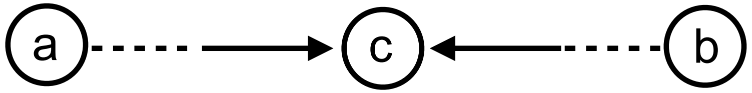
Two nodes a and b in a DAG are **d-separated** by a set of nodes Z if and only if *any* loop-free path from a to b satisfies one of the following:

1. 

Path contains a **chain** and c belongs to Z.

2. 

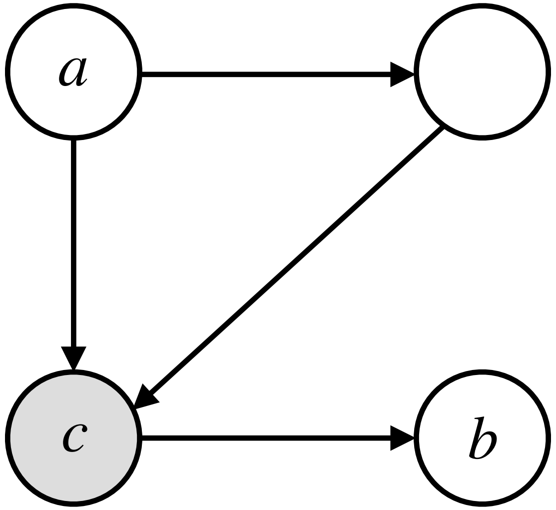
Path contains a **fork** and c belongs to Z.

3. 

Path contains a **collider** and c *does not* belong to Z.
In addition, no descendant of c belongs to Z.

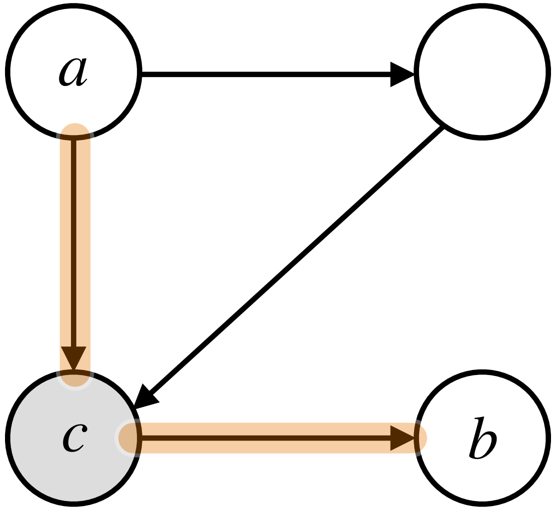
Property: variables a, b are independent given Z ⇔ they are d-separated by Z

Example of d-separation



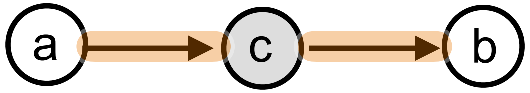
a and b are d-separated by c because

Example of d-separation

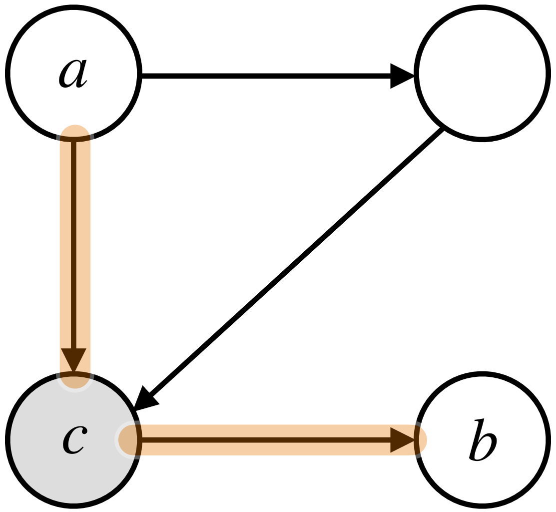


a and b are d-separated by c because

1. c is sandwiched by a **chain** in the path



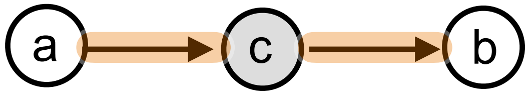
Example of d-separation



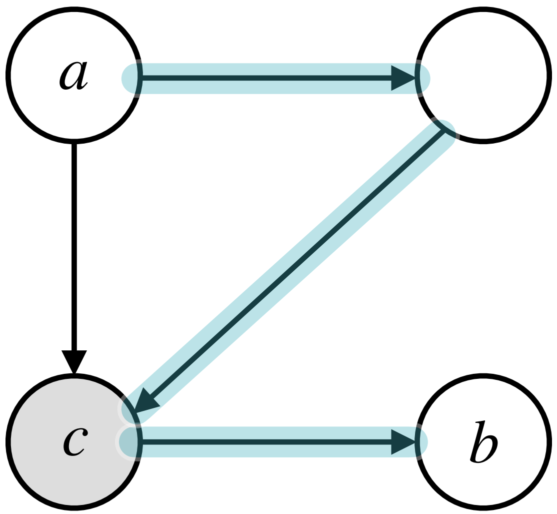
"Chain"

a and b are d-separated by c because

1. c is sandwiched by a **chain** in the path

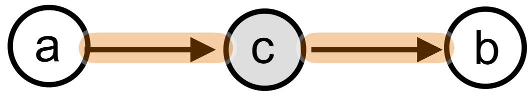


Example of d-separation

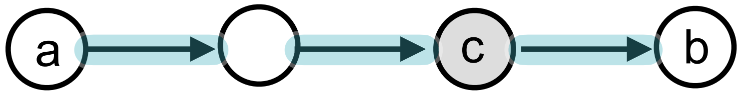


a and b are d-separated by c because

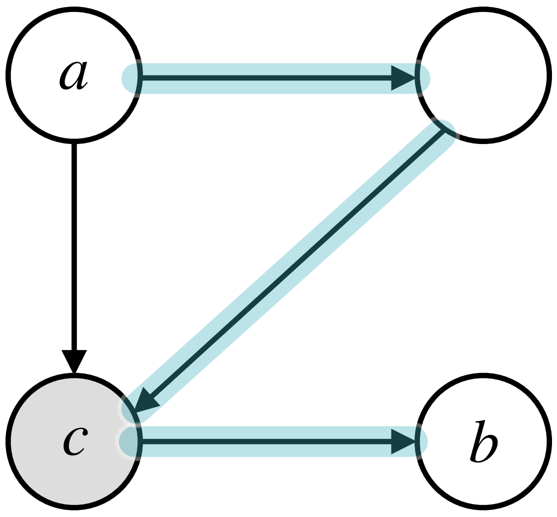
1. c is sandwiched by a **chain** in the path



2. c is sandwiched by a **chain** in the path



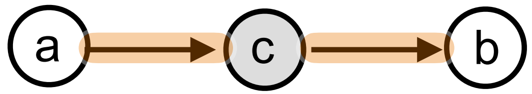
Example of d-separation



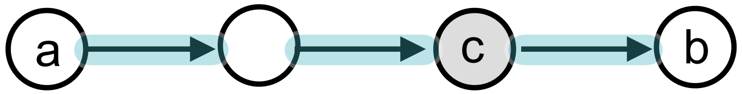
“Chain”

a and b are d-separated by c because

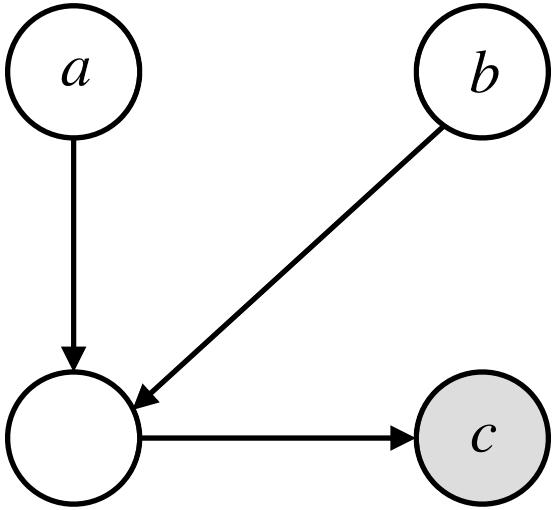
1. c is sandwiched by a **chain** in the path



2. c is sandwiched by a **chain** in the path



Non-example of d-separation

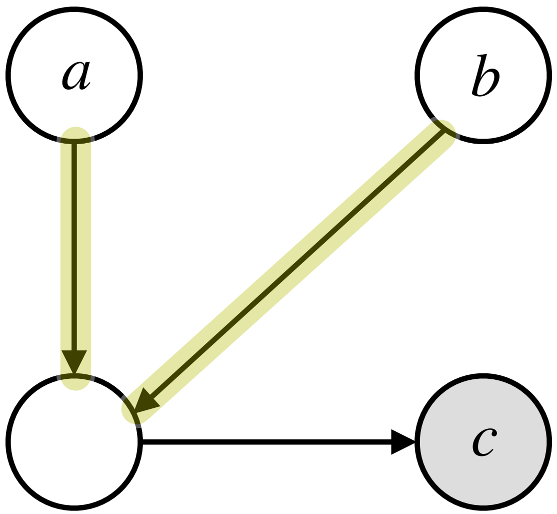


Nodes a and b are *not* d-separated by c

(i.e., a and b are d-connected)

because

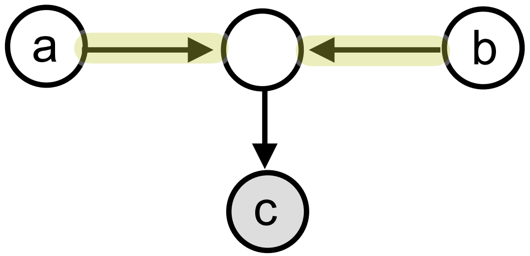
Non-example of d-separation



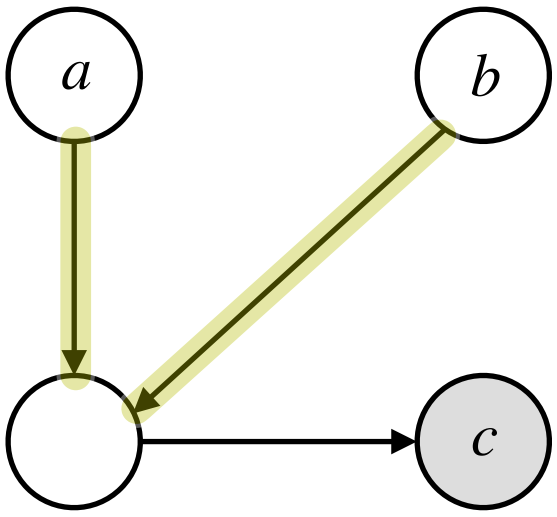
Nodes a and b are *not* d-separated by c

(i.e., a and b are d-connected)

because



Non-example of d-separation

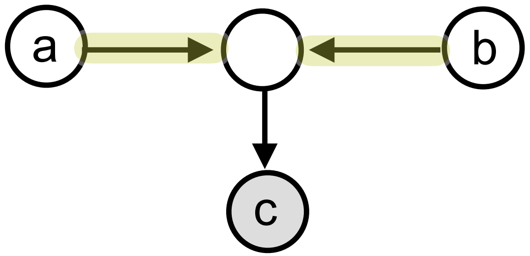


"Collider"

Nodes a and b are *not* d-separated by c

(i.e., a and b are d-connected)

because

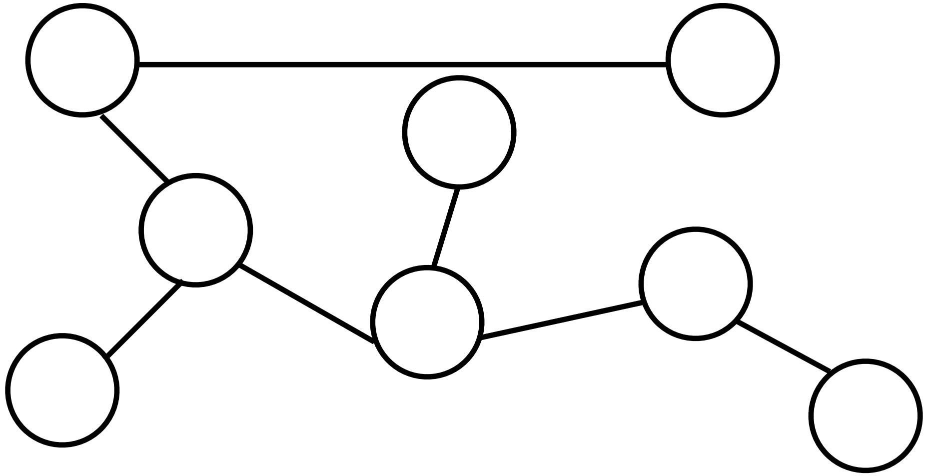


contains a **collider** and c is a descendant of the collider node

Markov Random Fields

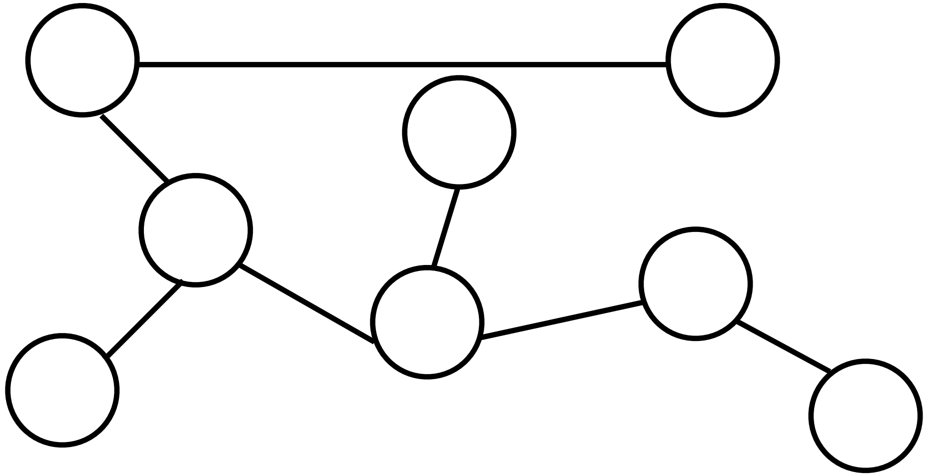
Markov Random Fields

- **Markov random fields (MRF)** are represented by **undirected graphs**



Markov Random Fields

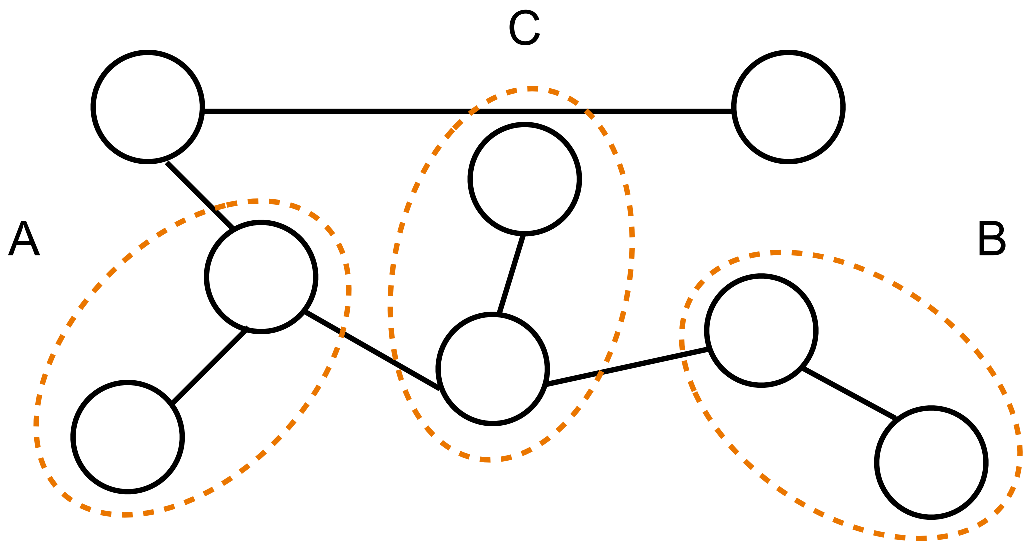
- **Markov random fields (MRF)** are represented by **undirected graphs**



- A and B are conditionally independent given C if and only if paths between points in A and B are **blocked** by C

Markov Random Fields

- **Markov random fields (MRF)** are represented by **undirected graphs**

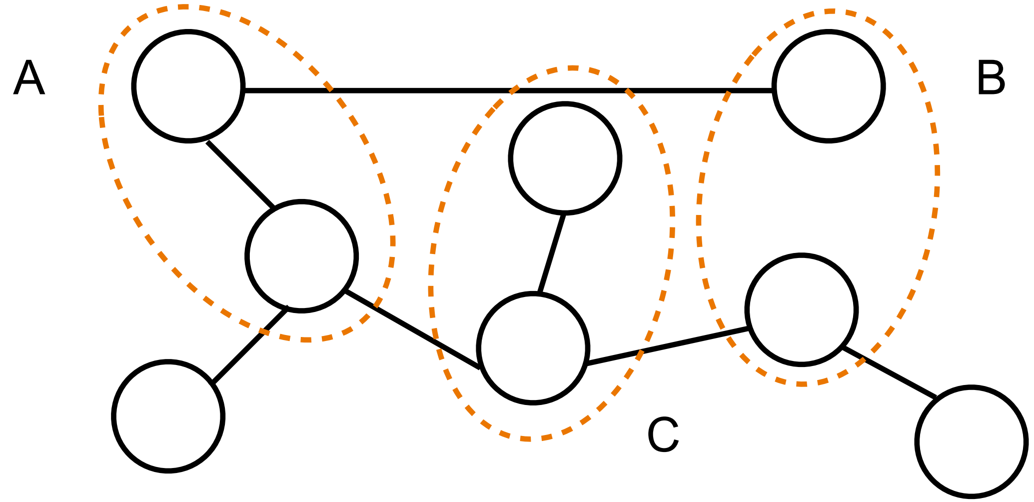


$$A \perp B \mid C$$

- A and B are conditionally independent given C if and only if paths between points in A and B are **blocked** by C

Markov Random Fields

- **Markov random fields (MRF)** are represented by **undirected graphs**

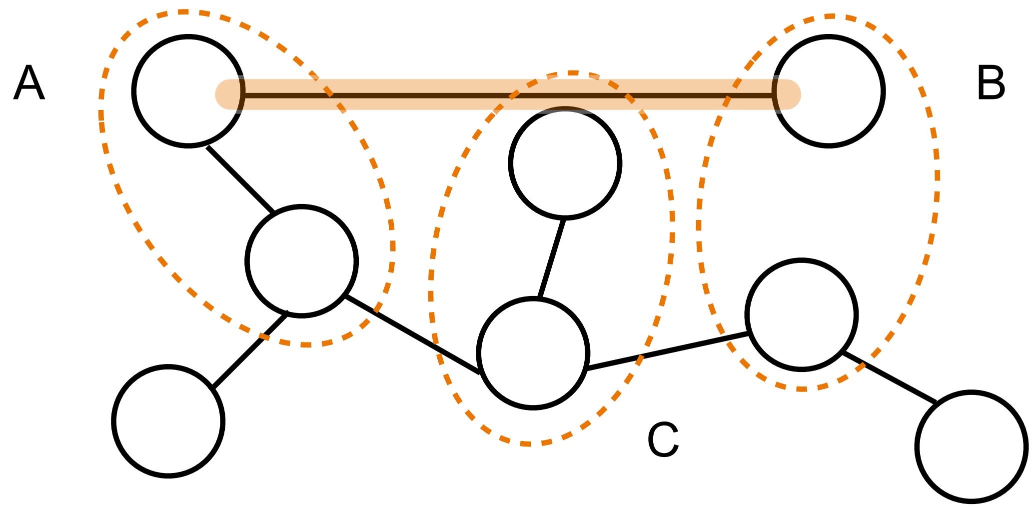


$A \not\perp B \mid C$

- A and B are conditionally independent given C if and only if paths between points in A and B are **blocked** by C

Markov Random Fields

- **Markov random fields (MRF)** are represented by **undirected graphs**

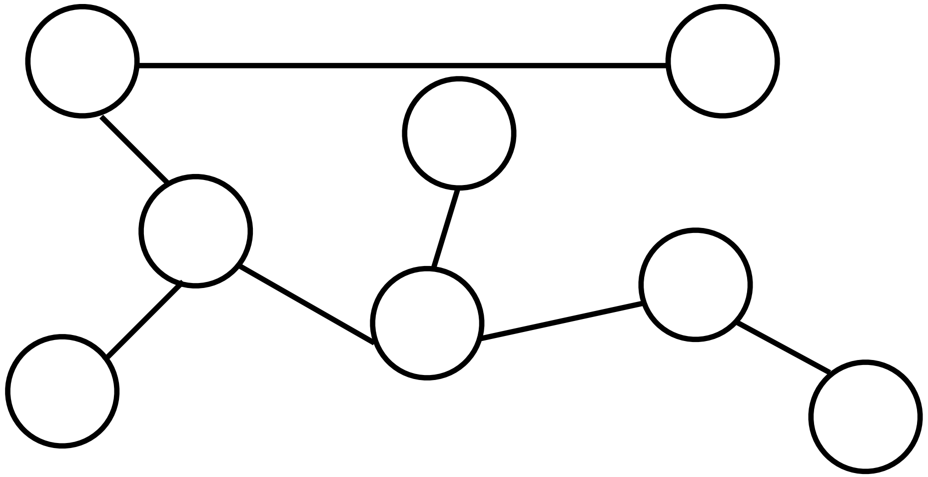


$A \not\perp B \mid C$

- A and B are conditionally independent given C if and only if paths between points in A and B are **blocked** by C

Markov Random Fields

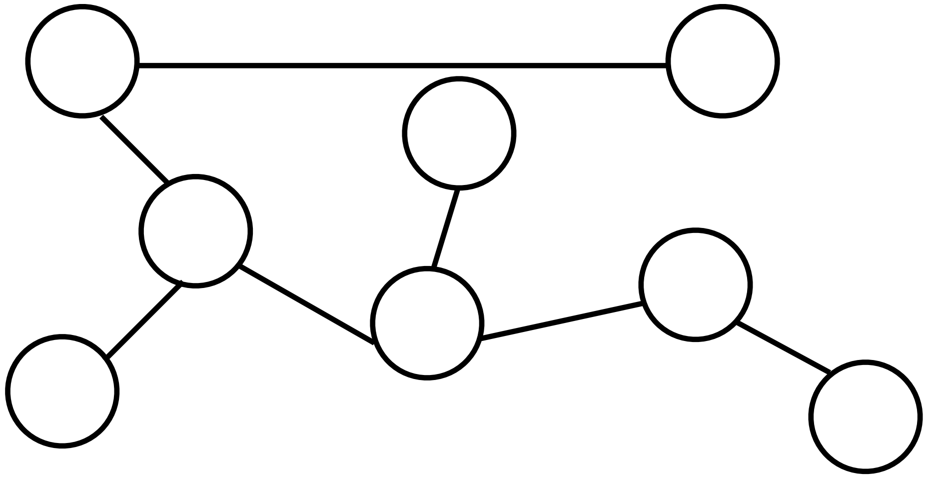
- **Markov random fields (MRF)** are represented by **undirected graphs**



- A and B are conditionally independent given C if and only if paths between points in A and B are **blocked** by C

Markov Random Fields

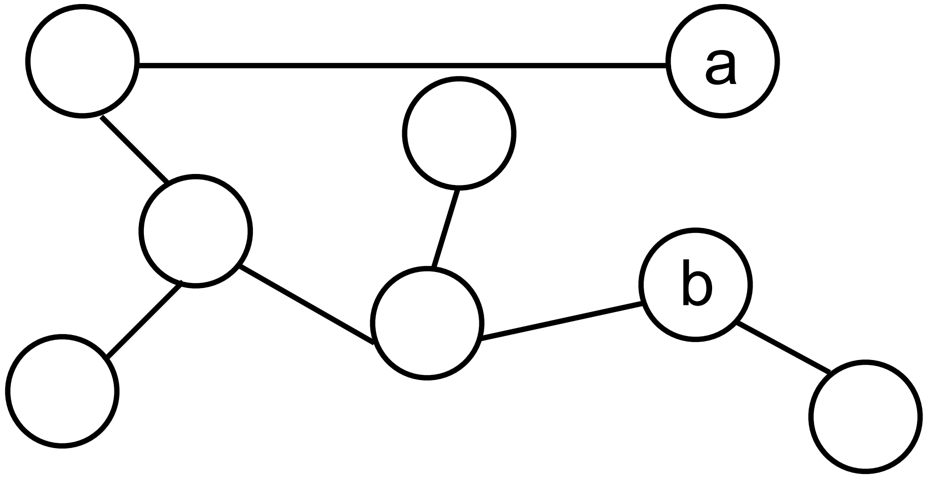
- **Markov random fields (MRF)** are represented by **undirected graphs**



- A and B are conditionally independent given C if and only if paths between points in A and B are **blocked** by C

Markov Random Fields

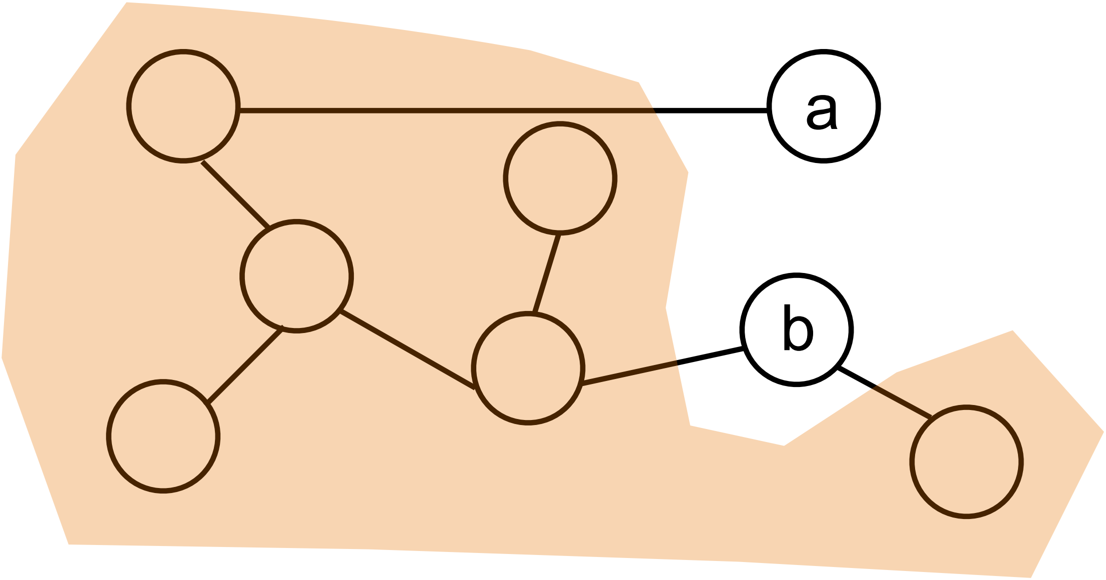
- Markov random fields (MRF) are represented by **undirected graphs**



- A and B are conditionally independent given C if and only if paths between points in A and B are **blocked** by C
- Thus, two nodes a and b are *non-adjacent* if and only if they are conditionally independent given all other nodes

Markov Random Fields

- **Markov random fields (MRF)** are represented by **undirected graphs**



- A and B are conditionally independent given C if and only if paths between points in A and B are **blocked** by C
- Thus, two nodes a and b are *non-adjacent* if and only if they are conditionally independent given all other nodes

Hammersley-Clifford Theorem

Hammersley-Clifford Theorem

In MRFs, we can consider factorisations into **potential functions** $\psi_C(\boldsymbol{x}_C) \geq 0$:

$$p(x_1, \ldots, x_n) \propto \prod_C \psi_C(\boldsymbol{x}_C),$$

where C is a clique of the graph*.

*Recall that a *clique* is a fully-connected subgraph of a graph

Hammersley-Clifford Theorem

In MRFs, we can consider factorisations into **potential functions** $\psi_C(\boldsymbol{x}_C) \geq 0$:

$$p(x_1, \ldots, x_n) \propto \prod_C \psi_C(\boldsymbol{x}_C),$$

where C is a clique of the graph*.

Akin to factorising **joint distributions** into *conditional distributions* in BNs.

*Recall that a *clique* is a fully-connected subgraph of a graph

Hammersley-Clifford Theorem

In MRFs, we can consider factorisations into **potential functions** $\psi_C(\boldsymbol{x}_C) \geq 0$:

$$p(x_1, \ldots, x_n) \propto \prod_C \psi_C(\boldsymbol{x}_C),$$

where C is a clique of the graph*.

Akin to factorising **joint distributions** into *conditional distributions* in BNs.

- Potential functions *need not* have a probabilistic interpretation

*Recall that a *clique* is a fully-connected subgraph of a graph

Hammersley-Clifford Theorem

In MRFs, we can consider factorisations into **potential functions** $\psi_C(\boldsymbol{x}_C) \geq 0$:

$$p(x_1, \ldots, x_n) \propto \prod_C \psi_C(\boldsymbol{x}_C),$$

where C is a clique of the graph*.

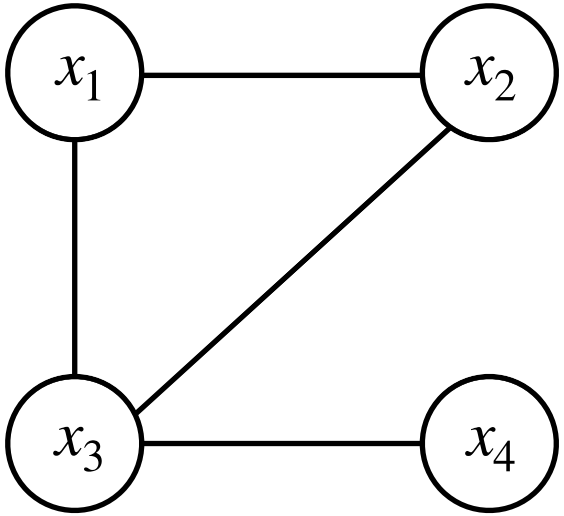
Akin to factorising **joint distributions** into *conditional distributions* in BNs.

- Potential functions *need not* have a probabilistic interpretation
- Factorisation is *not* unique

*Recall that a *clique* is a fully-connected subgraph of a graph

Example illustrating the Hammersley-Clifford theorem

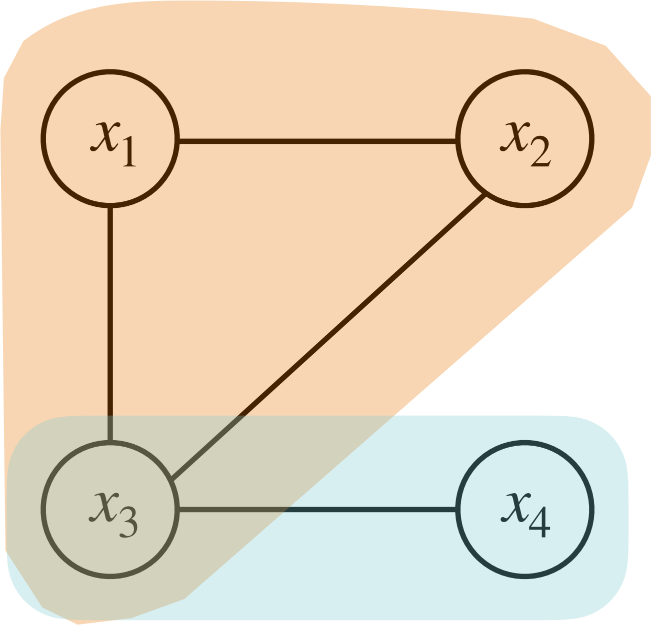
1. Factorisation into maximal cliques



$$p(x_1, x_2, x_3, x_4)$$

Example illustrating the Hammersley-Clifford theorem

1. Factorisation into maximal cliques

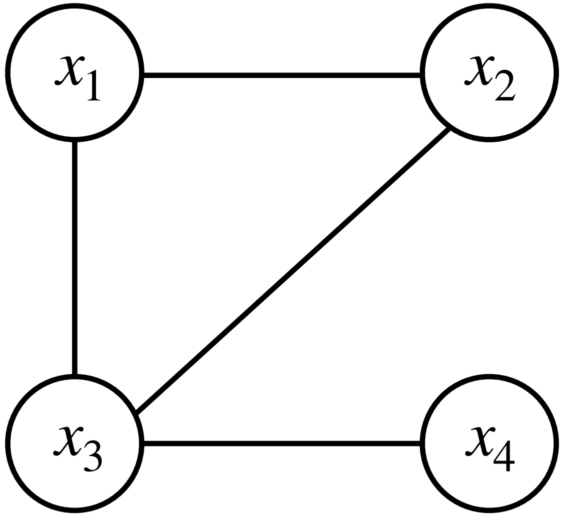


$$p(x_1, x_2, x_3, x_4)$$

$$= \psi_{123}(x_1, x_2, x_3)\,\psi_{34}(x_3, x_4)$$

Example illustrating the Hammersley-Clifford theorem

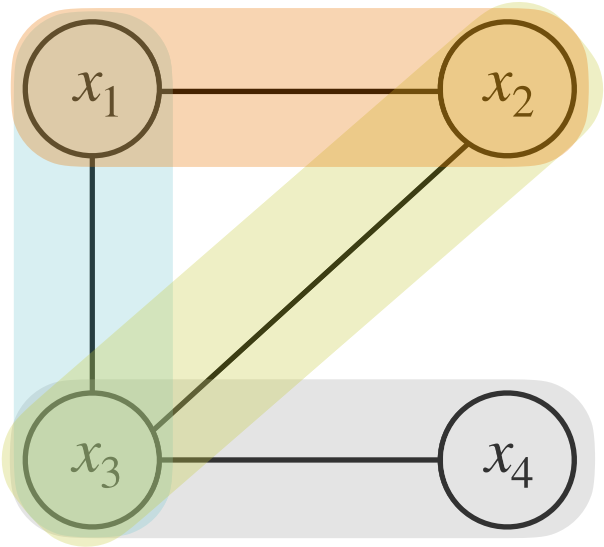
2. Factorisation into pairwise cliques



$$p(x_1, x_2, x_3, x_4)$$

Example illustrating the Hammersley-Clifford theorem

2. Factorisation into pairwise cliques

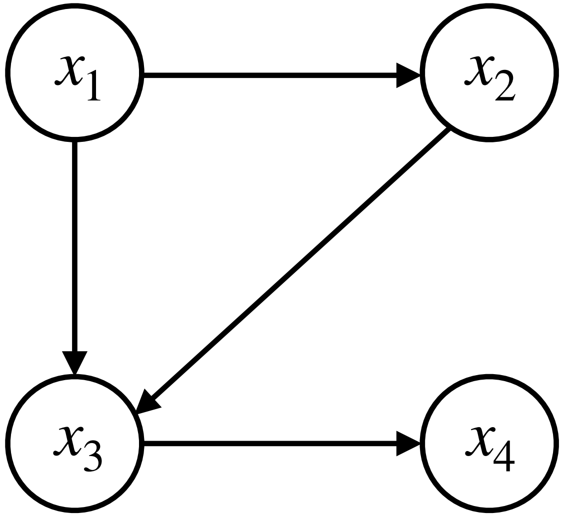


$$p(x_1, x_2, x_3, x_4)$$

$$= \psi_{12}(x_1, x_2)\,\psi_{13}(x_1, x_3)\,\psi_{23}(x_2, x_3)\,\psi_{34}(x_3, x_4)$$

Example illustrating the Hammersley-Clifford theorem

3. Factorisation of Bayesian networks

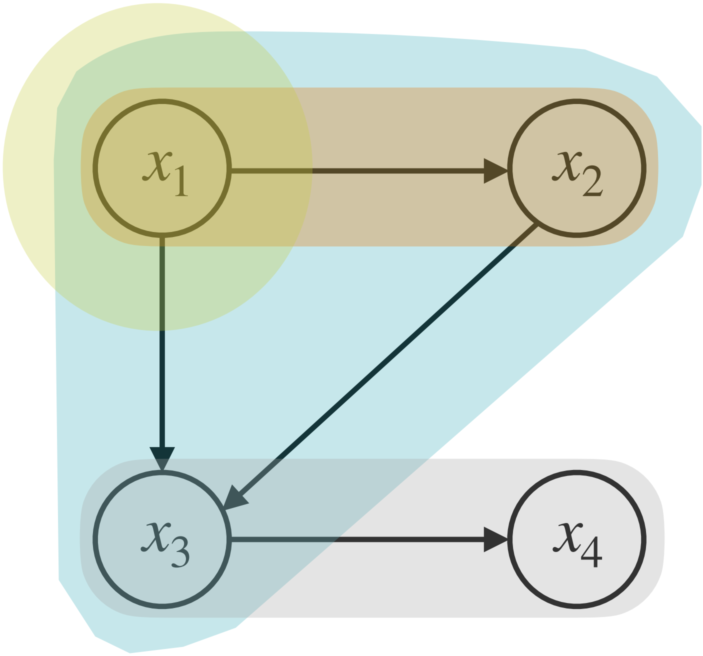


$$p(x_1, x_2, x_3, x_4)$$

$$= p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

Example illustrating the Hammersley-Clifford theorem

3. Factorisation of Bayesian networks



$$p(x_1, x_2, x_3, x_4)$$

$$= p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

$$= \psi_{34}(x_3, x_4)\, \psi_{123}(x_1, x_2, x_3)\, \psi_{12}(x_1, x_2)\, \psi_1(x_1)$$

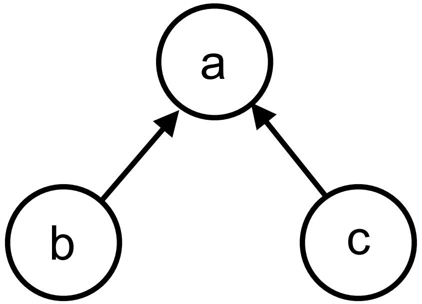
Markov Random Fields $\neq$ Bayesian Networks

Markov Random Fields $\neq$ Bayesian Networks

- MRFs *do not* represent BNs without altering the graph structure, e.g.

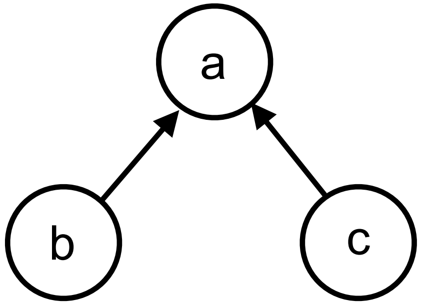
Markov Random Fields $\neq$ Bayesian Networks

- MRFs *do not* represent BNs without altering the graph structure, e.g.



Markov Random Fields $\neq$ Bayesian Networks

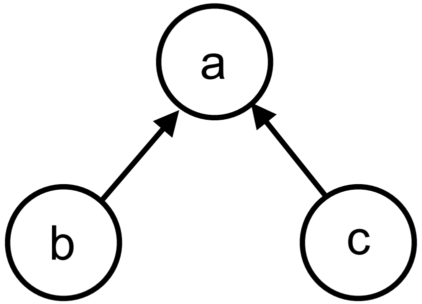
- MRFs *do not* represent BNs without altering the graph structure, e.g.



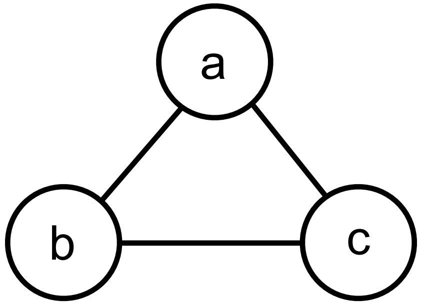
b and c are **not** conditionally independent given a

Markov Random Fields $\neq$ Bayesian Networks

- MRFs *do not* represent BNs without altering the graph structure, e.g.

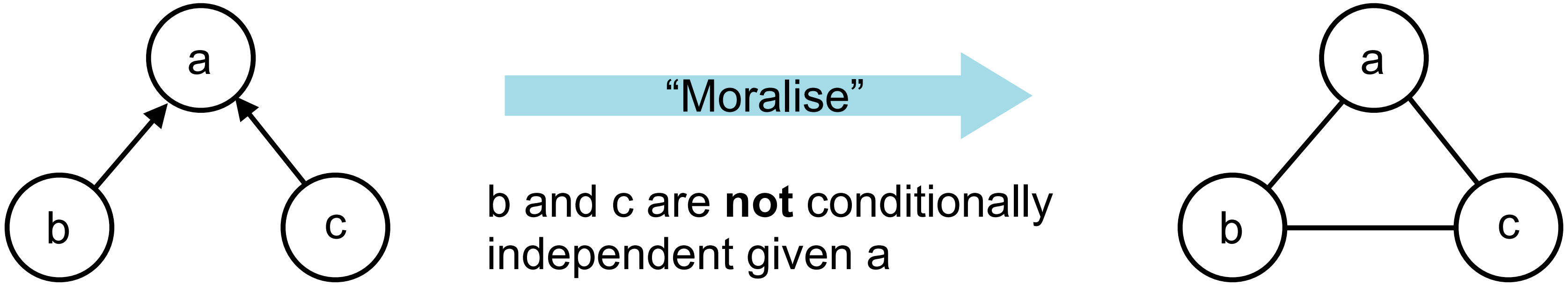


b and c are **not** conditionally independent given a



Markov Random Fields ≠ Bayesian Networks

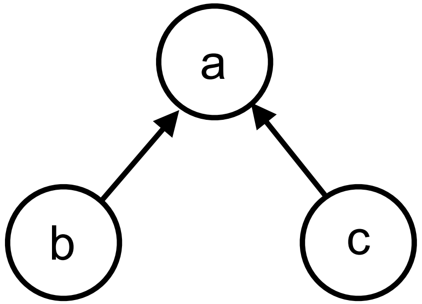
- MRFs *do not* represent BNs without altering the graph structure, e.g.



b and c are **not** conditionally independent given a

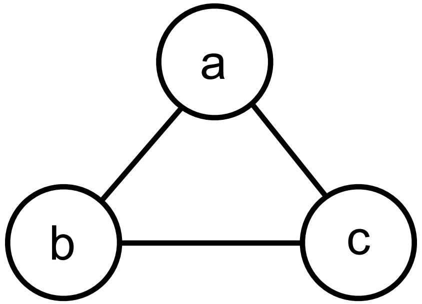
Markov Random Fields $\neq$ Bayesian Networks

- MRFs *do not* represent BNs without altering the graph structure, e.g.



“Moralise”

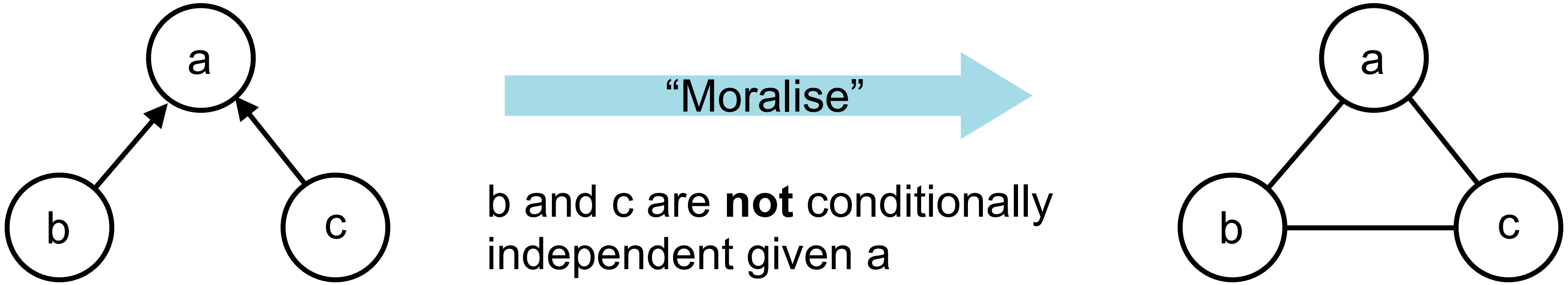
b and c are **not** conditionally independent given a



- Not all MRFs can be represented as a BN, e.g.

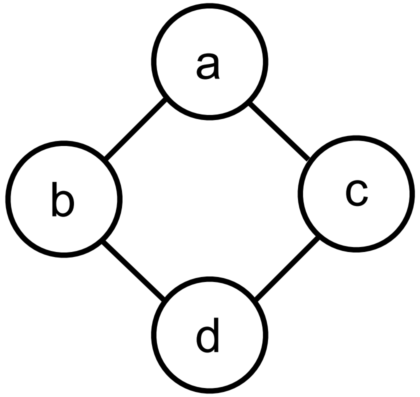
Markov Random Fields $\neq$ Bayesian Networks

- MRFs *do not* represent BNs without altering the graph structure, e.g.



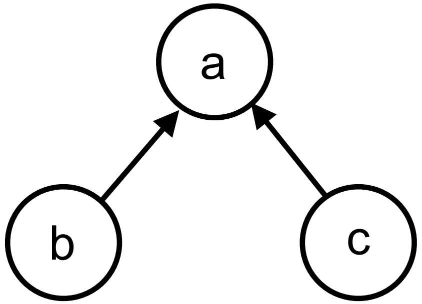
b and c are **not** conditionally independent given a

- Not all MRFs can be represented as a BN, e.g.

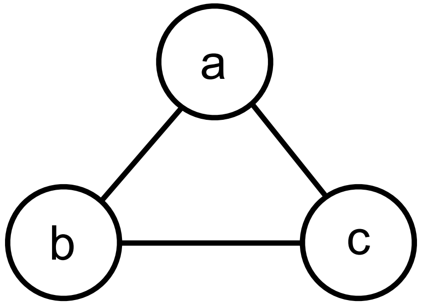


Markov Random Fields ≠ Bayesian Networks

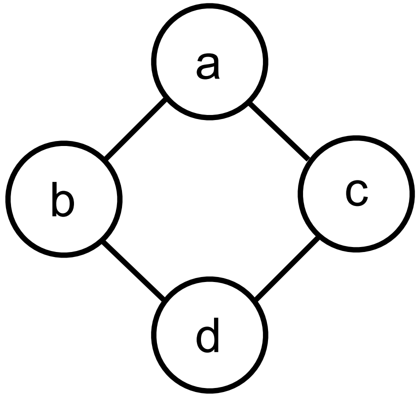
- MRFs *do not* represent BNs without altering the graph structure, e.g.



b and c are **not** conditionally independent given a



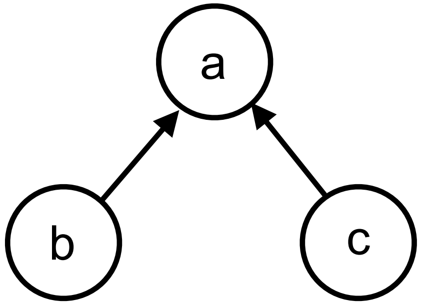
- Not all MRFs can be represented as a BN, e.g.



{a, d} are conditionally independent given {b, c} and vice-versa.

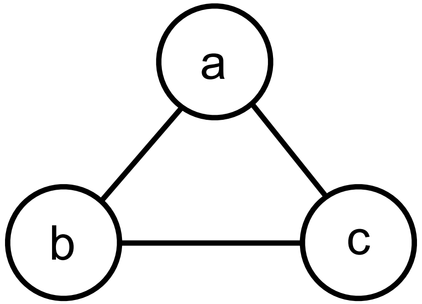
Markov Random Fields $\neq$ Bayesian Networks

- MRFs *do not* represent BNs without altering the graph structure, e.g.

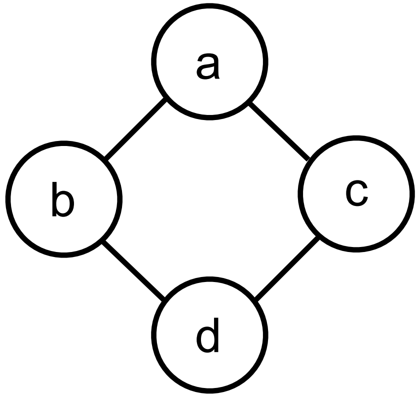


“Moralise”

b and c are **not** conditionally independent given a



- Not all MRFs can be represented as a BN, e.g.



{a, d} are conditionally independent given {b, c} and vice-versa.
Cannot happen in a DAG.

Factor Graphs

Factor Graphs

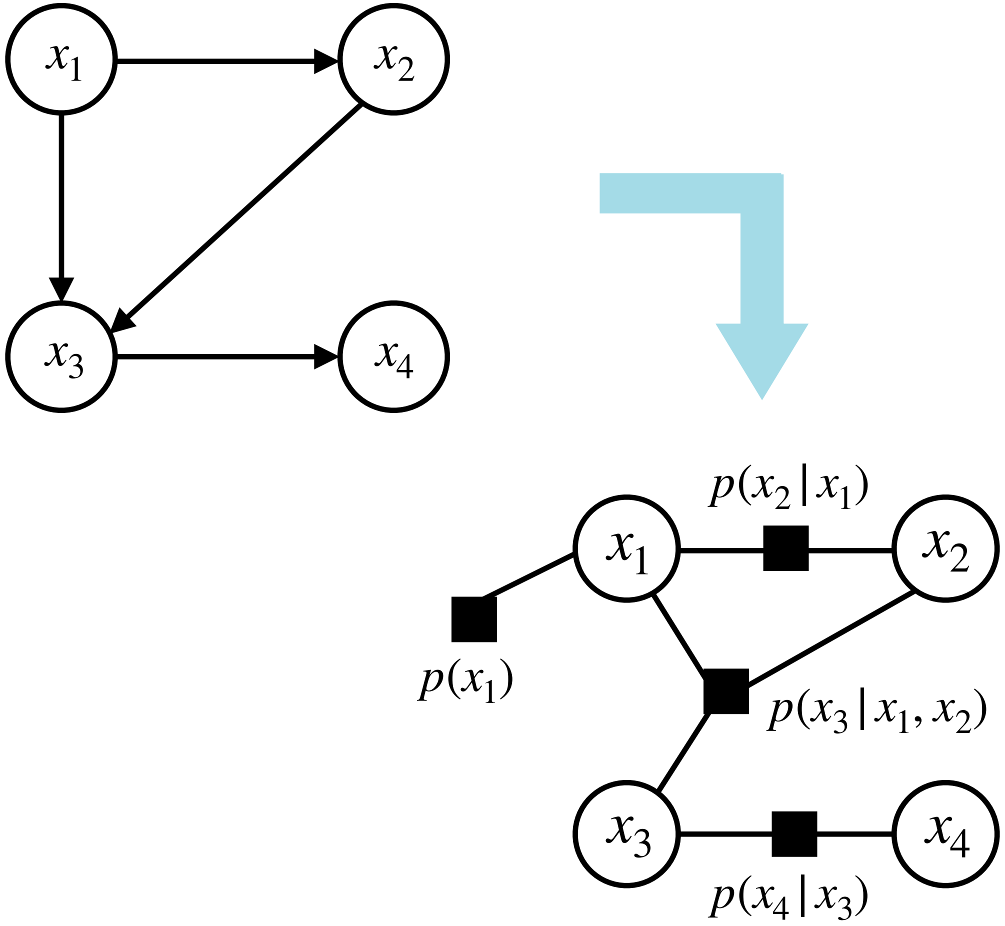
- **Factor graphs** are alternative representations of BNs and MRFs

Factor Graphs

- **Factor graphs** are alternative representations of BNs and MRFs
- Makes the factorisation explicit

Factor Graphs

$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

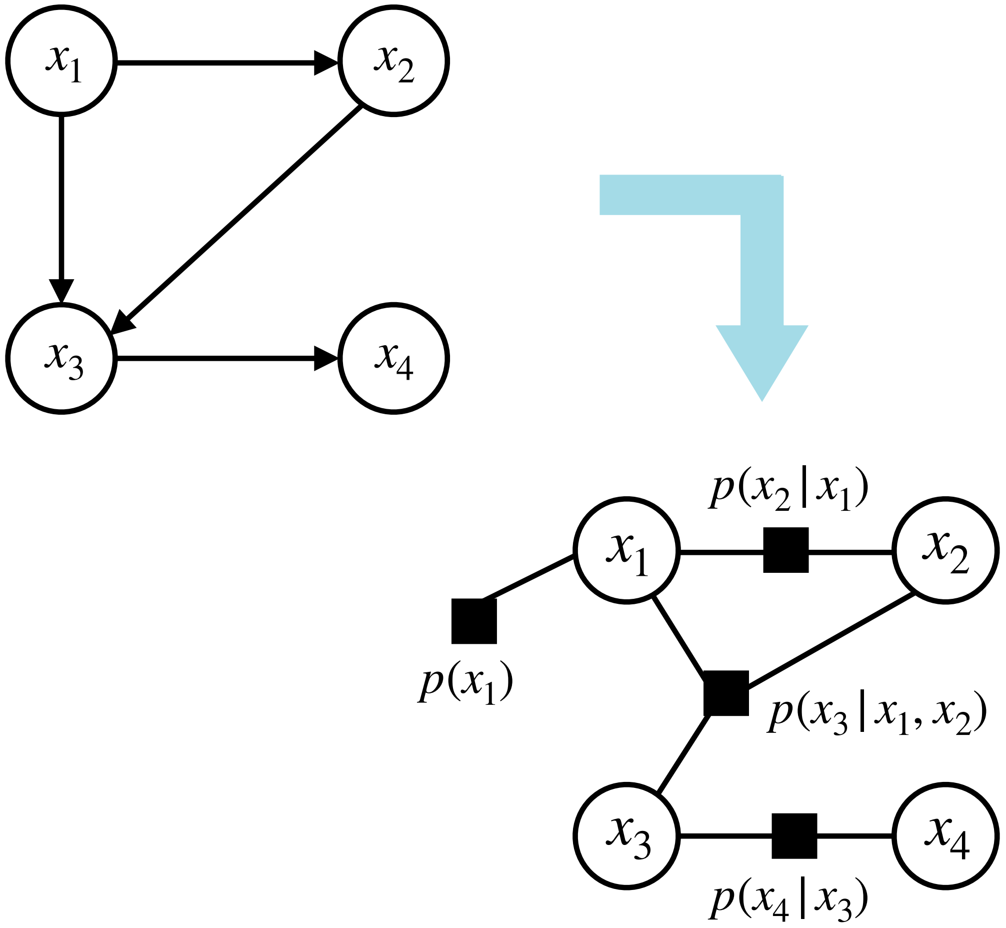


- **Factor graphs** are alternative representations of BNs and MRFs

- Makes the factorisation explicit

Factor Graphs

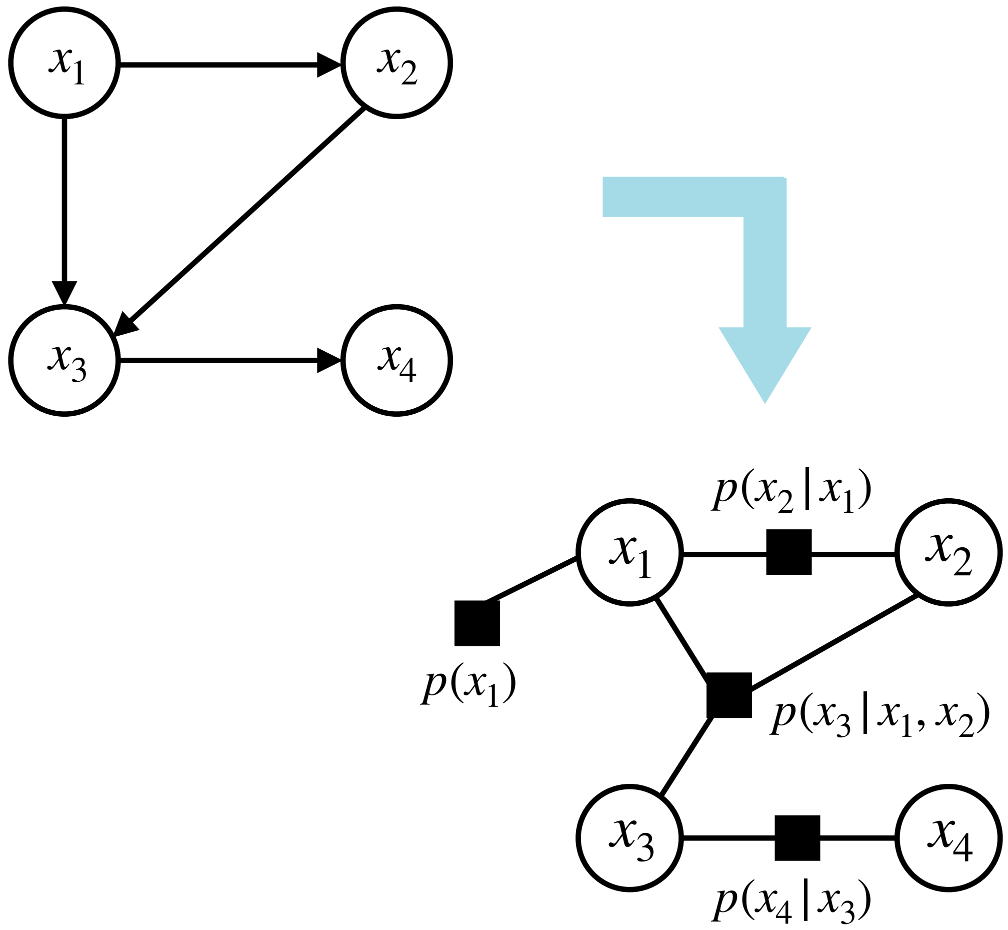
$$p(x_1, x_2, x_3, x_4) = p(x_4 | x_3)\, p(x_3 | x_1, x_2)\, p(x_2 | x_1)\, p(x_1)$$



- **Factor graphs** are alternative representations of BNs and MRFs
- Makes the factorisation explicit
- Circle nodes (○) represent variables

Factor Graphs

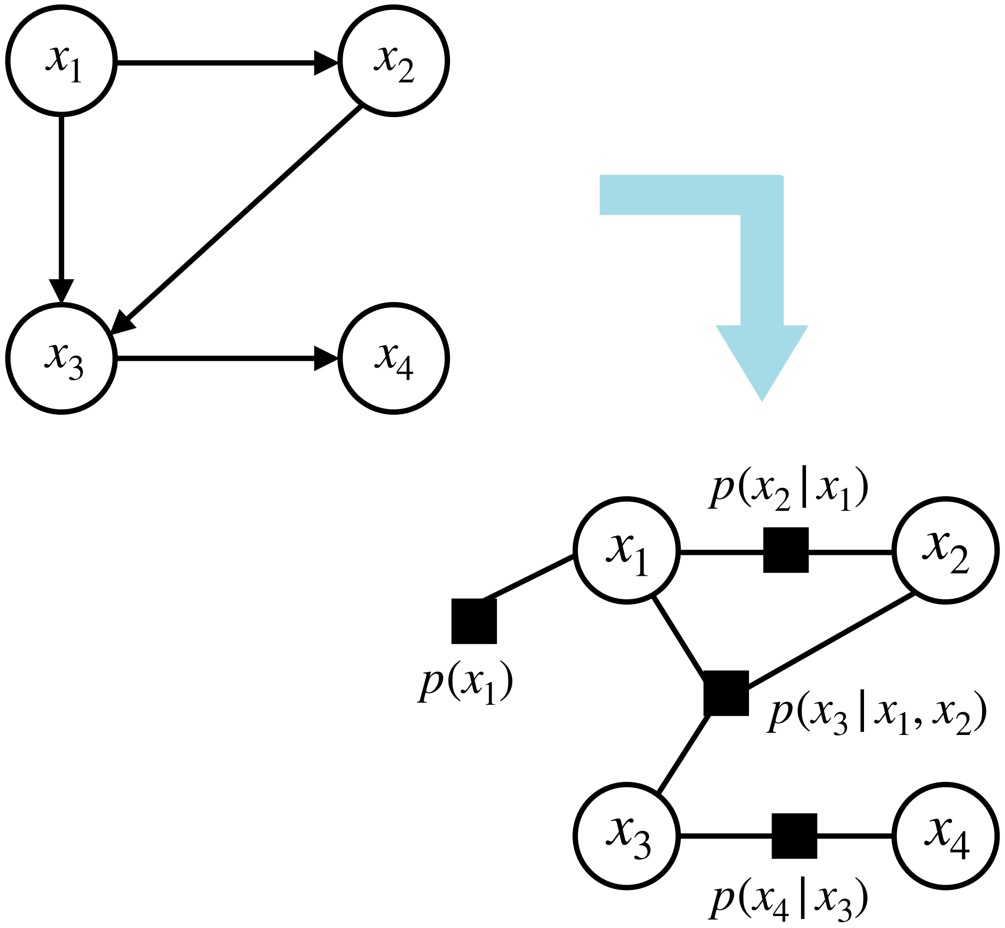
$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$



- **Factor graphs** are alternative representations of BNs and MRFs
- Makes the factorisation explicit
- Circle nodes (○) represent variables
- Square nodes (■) represent *factors*

Factor Graphs

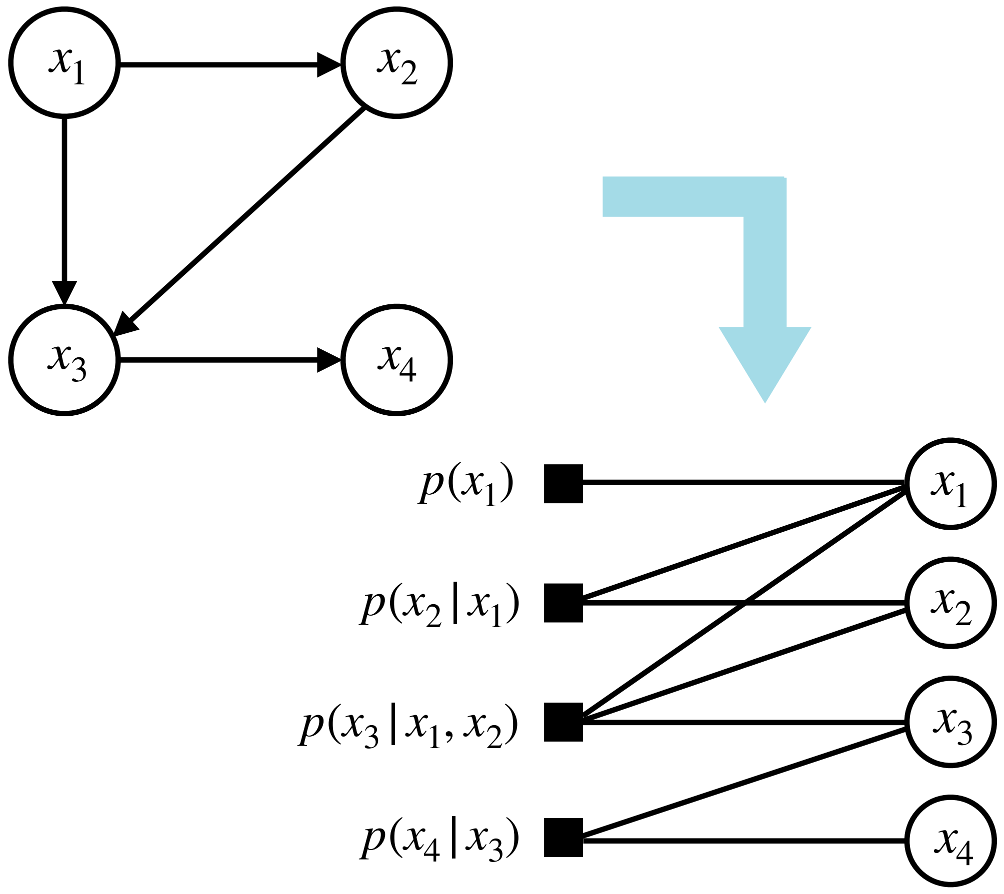
$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$



- **Factor graphs** are alternative representations of BNs and MRFs
- Makes the factorisation explicit
- Circle nodes (○) represent variables
- Square nodes (■) represent *factors*
- Graph is **undirected** and **bipartite**

Factor Graphs

$$p(x_1, x_2, x_3, x_4) = p(x_4 \mid x_3)\, p(x_3 \mid x_1, x_2)\, p(x_2 \mid x_1)\, p(x_1)$$

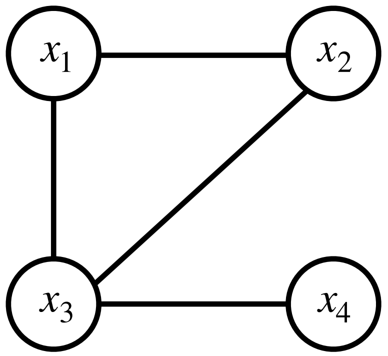


- **Factor graphs** are alternative representations of BNs and MRFs
- Makes the factorisation explicit
- Circle nodes (○) represent variables
- Square nodes (■) represent *factors*
- Graph is **undirected** and **bipartite**

Factor Graphs

Factor graphs make the factorisation explicit

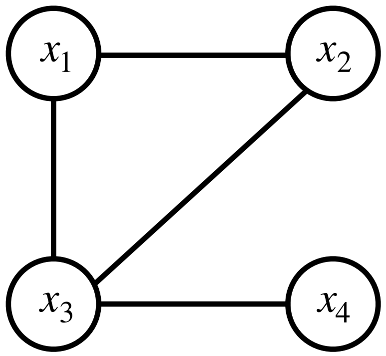
$\Rightarrow$ Useful for MRFs where factorisation is non-unique



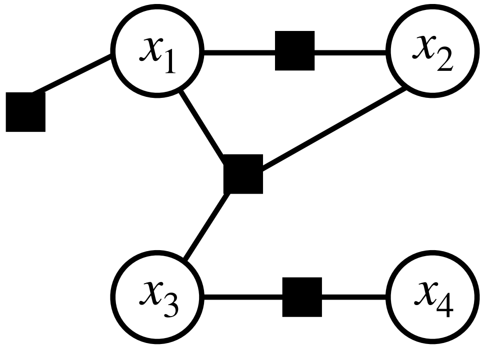
Factor Graphs

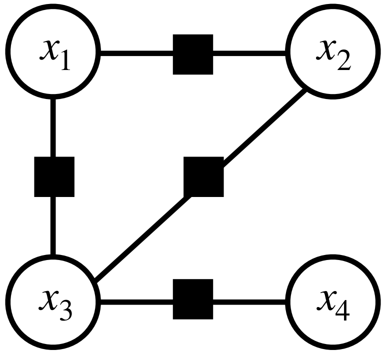
Factor graphs make the factorisation explicit

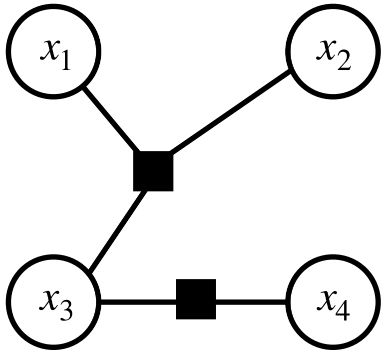
⇒ Useful for MRFs where factorisation is non-unique



There are many ways of factorising into potentials:





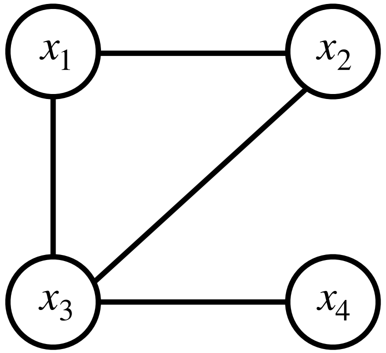


$\psi_1(x_1)\psi_{12}(x_1, x_2)\psi_{123}(x_1, x_2, x_3)\psi_{34}(x_3, x_4)$

$\psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)\psi_{13}(x_1, x_3)\psi_{34}(x_3, x_4)$

$\psi_{123}(x_1, x_2, x_3)\psi_{34}(x_3, x_4)$

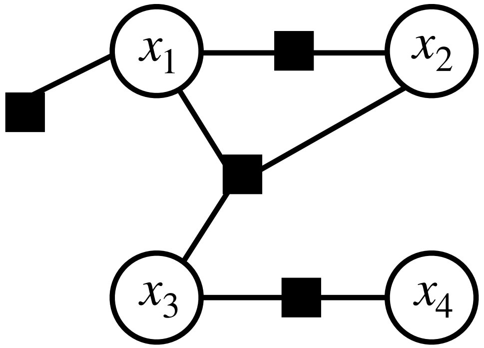
Factor Graphs

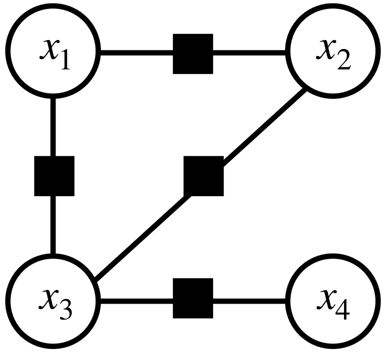


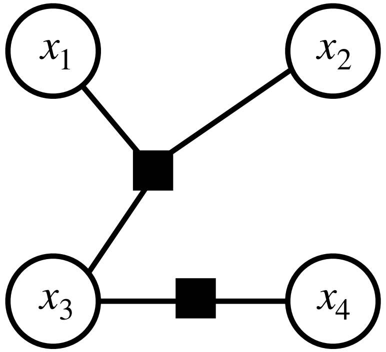
Factor graphs make the factorisation explicit

$\Rightarrow$ Useful for MRFs where factorisation is non-unique

There are many ways of factorising into potentials:





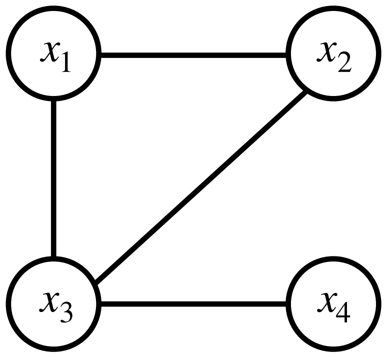


$\psi_1(x_1)\psi_{12}(x_1, x_2)\psi_{123}(x_1, x_2, x_3)\psi_{34}(x_3, x_4)$

$\psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)\psi_{13}(x_1, x_3)\psi_{34}(x_3, x_4)$

$\psi_{123}(x_1, x_2, x_3)\psi_{34}(x_3, x_4)$

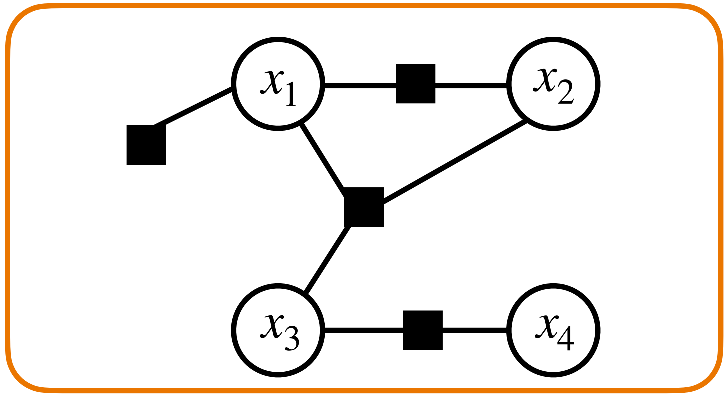
Factor Graphs

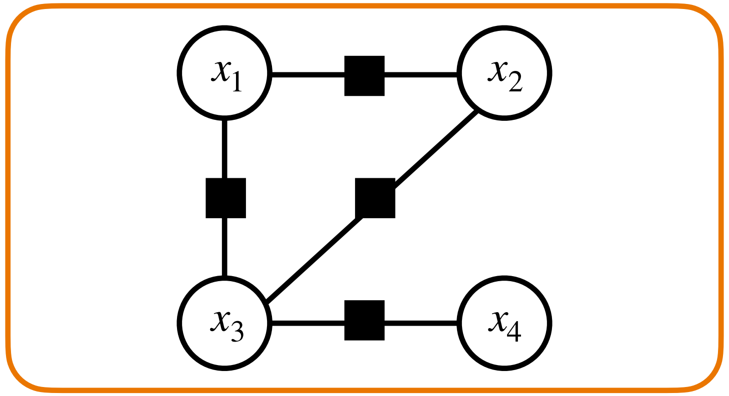


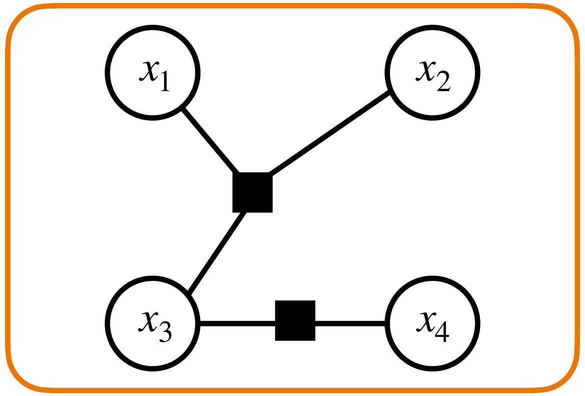
Factor graphs make the factorisation explicit

$\Rightarrow$ Useful for MRFs where factorisation is non-unique

There are many ways of factorising into potentials:







$\psi_1(x_1)\psi_{12}(x_1, x_2)\psi_{123}(x_1, x_2, x_3)\psi_{34}(x_3, x_4)$

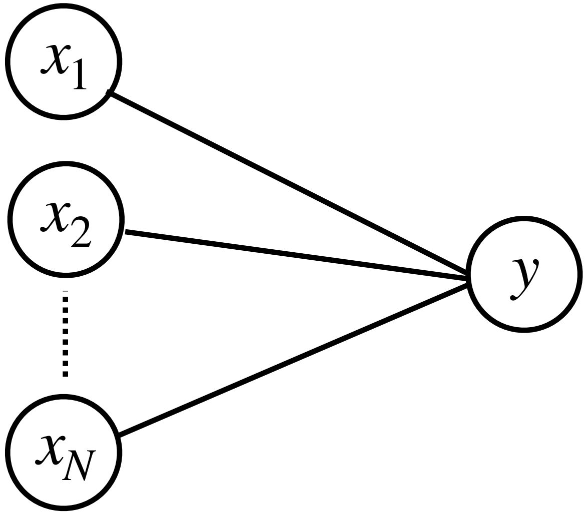
$\psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)\psi_{13}(x_1, x_3)\psi_{34}(x_3, x_4)$

$\psi_{123}(x_1, x_2, x_3)\psi_{34}(x_3, x_4)$

Useful notations

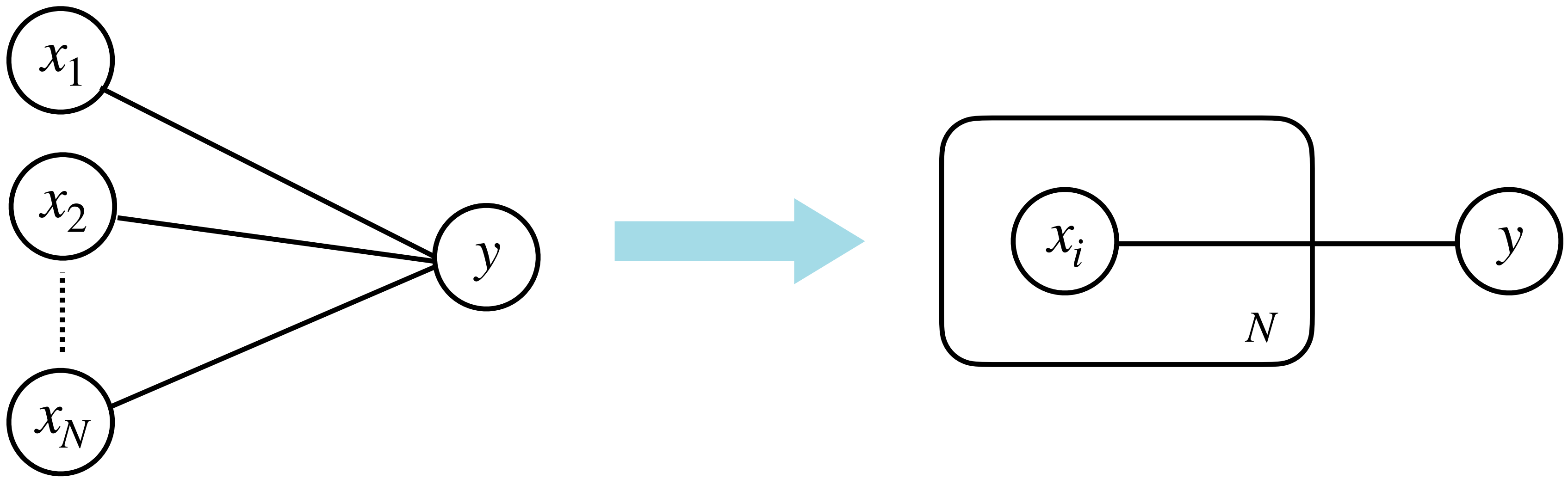
Useful notations

- Plate notation



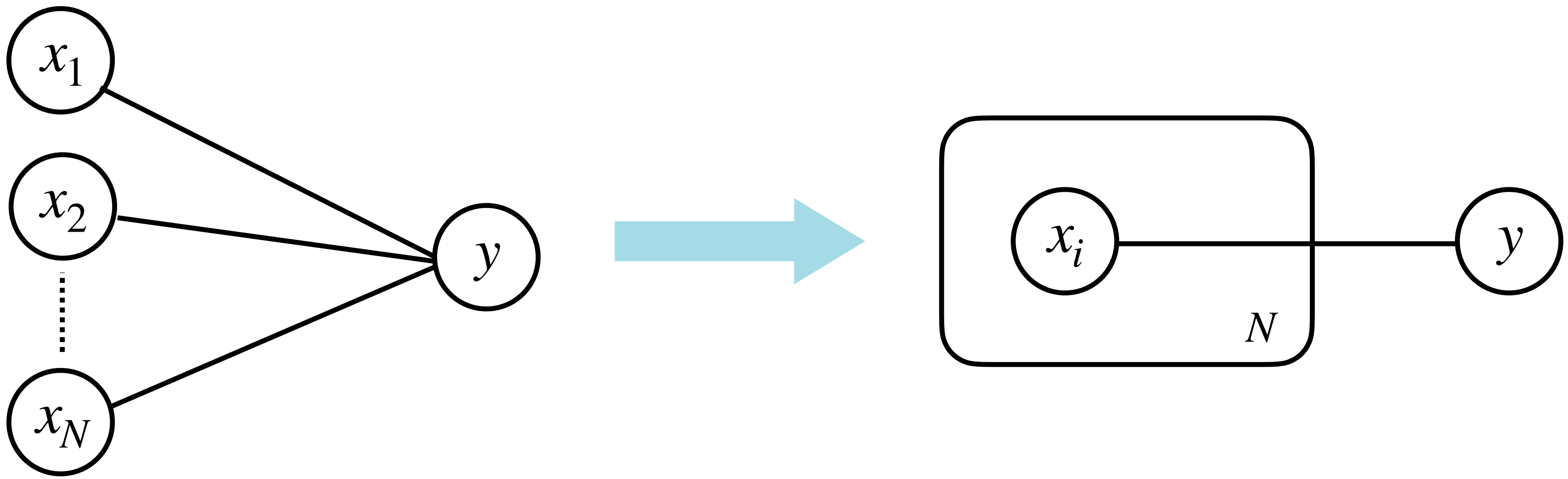
Useful notations

- Plate notation

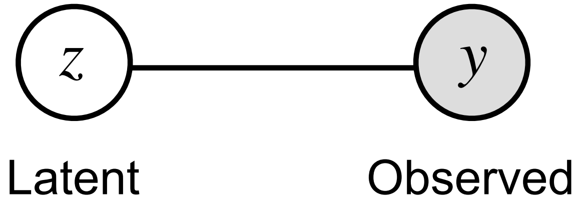


Useful notations

- Plate notation

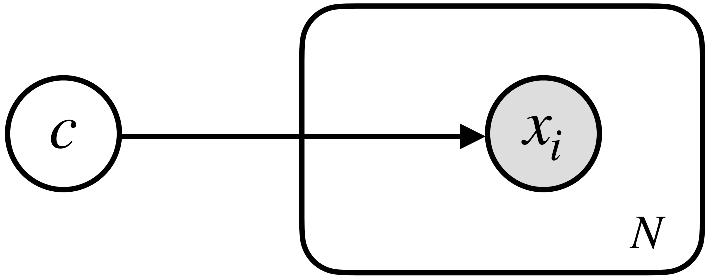


- Shaded vs. unshaded nodes

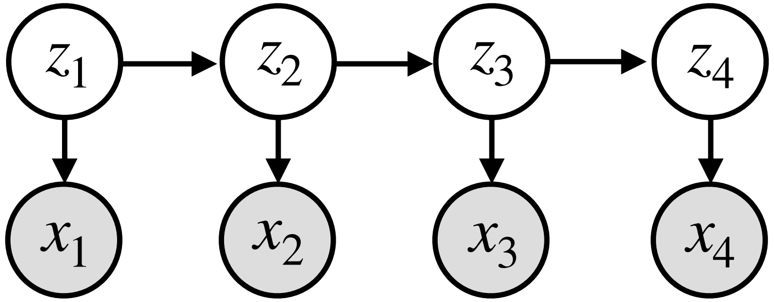


Examples of Bayesian networks

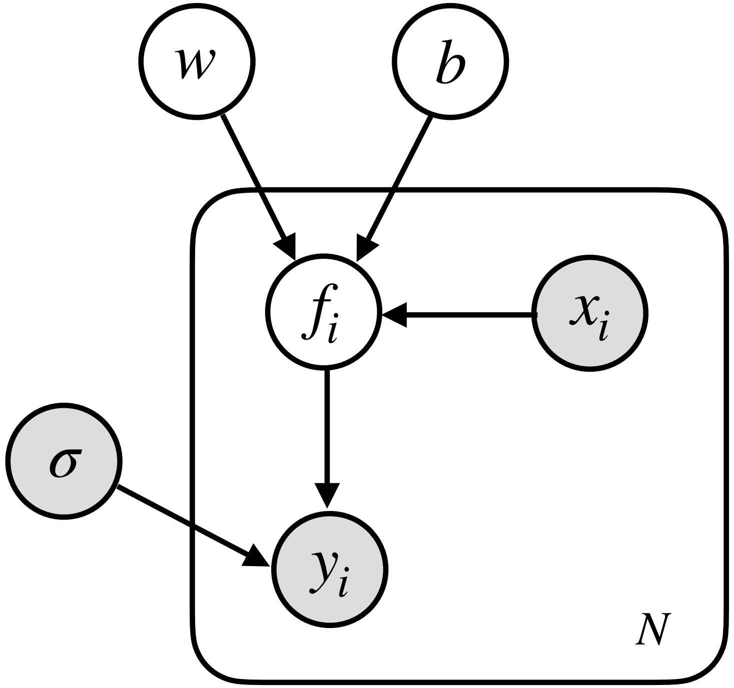
- Naive Bayes classifier



- Hidden Markov model



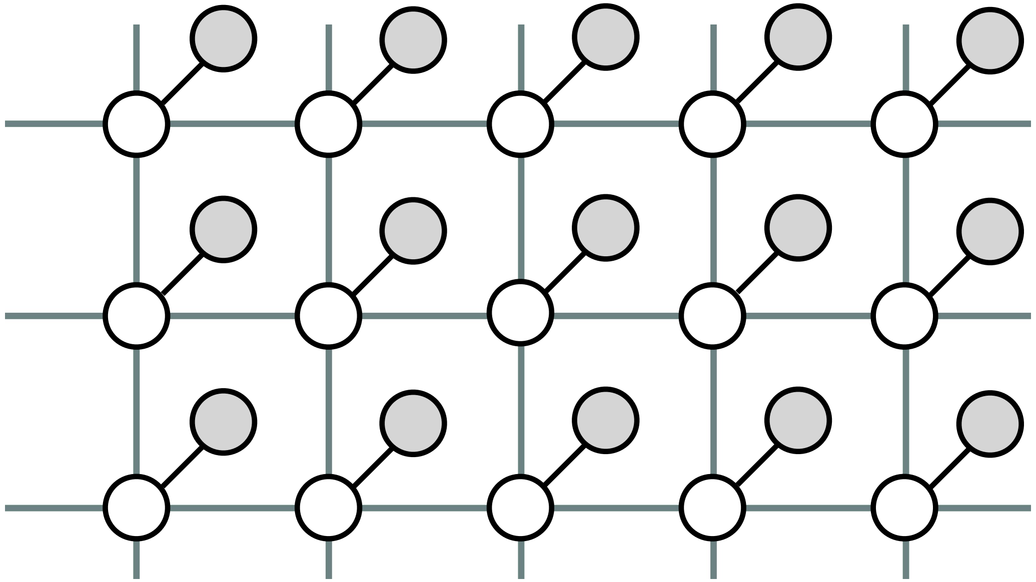
- Bayesian linear regression



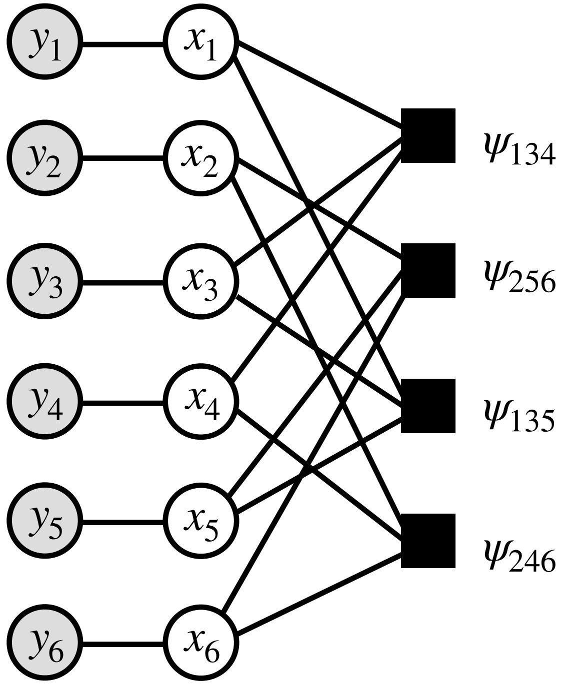
$$y_i = f_i + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma),$$
$$f_i = w x_i + b\,.$$

Examples of Markov random fields

- Spatial analysis / image processing [3,4]



- Error-correcting codes [5]



References

[1] Bishop, Christopher M. *Pattern Recognition and Machine Learning*. New York: springer, 2006.

[2] Koller, Daphne, and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

[3] Besag, Julian. *Spatial interaction and the statistical analysis of lattice systems*. Journal of the Royal Statistical Society: Series B (Methodological). 1974.

[4] Besag, Julian. *On the statistical analysis of dirty pictures.* Journal of the Royal Statistical Society: Series B (Methodological). 1986.

[5] Gallager, Robert. *Low-density parity-check codes.* IRE Transactions on information theory. 1962.

2. Belief Propagation on PGMs

Statistical inference with PGMs

In Bayesian statistics, we often need to compute:

1. The marginal likelihood $p(y)$ of observed data
2. The marginal distribution $p(z)$ of latent variables
3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$
4. The mode $\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Statistical inference with PGMs

In Bayesian statistics, we often need to compute:

- 1. The marginal likelihood $p(y)$ of observed data

2. The marginal distribution $p(z)$ of latent variables

3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$

4. The mode $\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Statistical inference with PGMs

In Bayesian statistics, we often need to compute:

1. The marginal likelihood $p(y)$ of observed data
2. The marginal distribution $p(z)$ of latent variables
3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$
4. The mode $\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Statistical inference with PGMs

In Bayesian statistics, we often need to compute:

1. The marginal likelihood $p(y)$ of observed data
2. The marginal distribution $p(z)$ of latent variables
3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$
4. The mode $\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Statistical inference with PGMs

In Bayesian statistics, we often need to compute:

1. The marginal likelihood $p(y)$ of observed data
2. The marginal distribution $p(z)$ of latent variables
3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$
4. The mode $\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Statistical inference with PGMs

In Bayesian statistics, we often need to compute:

1. The marginal likelihood $p(y)$ of observed data
2. The marginal distribution $p(z)$ of latent variables
3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$
4. The mode $\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Statistical inference with PGMs

In Bayesian statistics, we often need to compute:

1. The marginal likelihood $p(y)$ of observed data
2. The marginal distribution $p(z)$ of latent variables
3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$
4. The mode $\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Here, we will focus on computing *marginal distributions* using PGMs.

Statistical inference with PGMs

In Bayesian statistics, we often need to compute:

- 1. The marginal likelihood $p(y)$ of observed data
- 2. The marginal distribution $p(z)$ of latent variables
- 3. The conditional distribution $p(x_i \,|\, x_j)$ for any $i, j \in V$

4. The mode $\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Here, we will focus on computing *marginal distributions* using PGMs.

Example

Let $X_1, \ldots, X_N$ be random variables, each with K discrete states and p is the joint probability mass function.

Example

Let $X_1, \ldots, X_N$ be random variables, each with K discrete states and p is the joint probability mass function.

Question: What is the *marginal distribution* $p(X_i = x_i)$ for all $i = 1, \ldots, N$?

Example

Let $X_1, \ldots, X_N$ be random variables, each with K discrete states and p is the joint probability mass function.

Question: What is the *marginal distribution* $p(X_i = x_i)$ for all $i = 1,\ldots,N$?

A naive solution:

$$p(X_i = x_i) = \sum_{j \neq i}^{N} \left(\sum_{x_j \in \{1,\ldots,K\}} p(x_1, \ldots, x_N) \right).$$

Example

Let $X_1, \ldots, X_N$ be random variables, each with K discrete states and p is the joint probability mass function.

Question: What is the *marginal distribution* $p(X_i = x_i)$ for all $i = 1,\ldots,N$?

A naive solution:

$$p(X_i = x_i) = \sum_{j \neq i}^{N} \left(\sum_{x_j \in \{1,\ldots,K\}} p(x_1, \ldots, x_N) \right).$$

This has computational cost $\mathcal{O}(K^N)$

Example

Let $X_1, \ldots, X_N$ be random variables, each with K discrete states and p is the joint probability mass function.

Question: What is the *marginal distribution* $p(X_i = x_i)$ for all $i = 1, \ldots, N$?

A naive solution:

$$p(X_i = x_i) = \sum_{j \neq i}^{N} \left(\sum_{x_j \in \{1, \ldots, K\}} p(x_1, \ldots, x_N) \right).$$

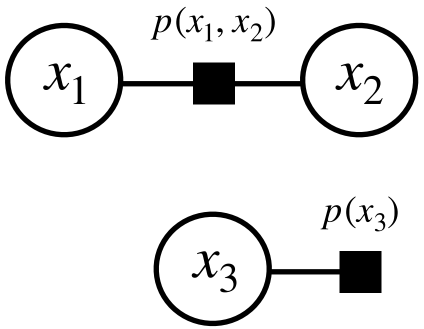
This has computational cost $\mathcal{O}(K^N)$ 👎

Assuming independence

Let $N = 3$ and assume we can write $p(x_1, x_2, x_3) = p(x_1, x_2)p(x_3)$.

Assuming independence

Let $N = 3$ and assume we can write $p(x_1, x_2, x_3) = p(x_1, x_2)p(x_3)$.

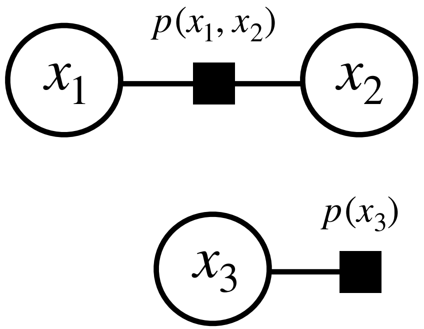


Assuming independence

Let $N = 3$ and assume we can write $p(x_1, x_2, x_3) = p(x_1, x_2)p(x_3)$.

Then, we have

$$p(x_1) = \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3)$$

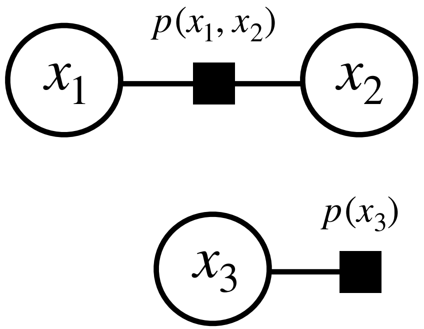


Assuming independence

Let $N = 3$ and assume we can write $p(x_1, x_2, x_3) = p(x_1, x_2)p(x_3)$.

Then, we have

$$p(x_1) = \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3)$$
$$= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2)p(x_3)$$

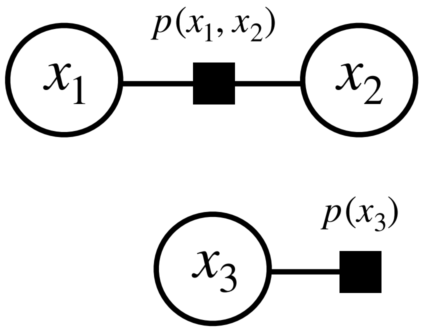


Assuming independence

Let $N = 3$ and assume we can write $p(x_1, x_2, x_3) = p(x_1, x_2)p(x_3)$.

Then, we have

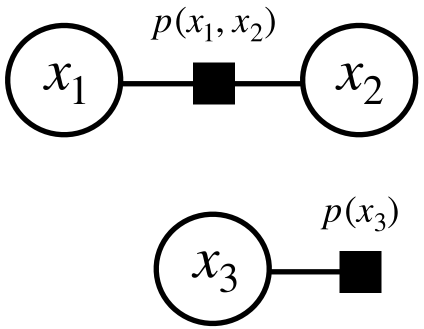
$$
\begin{aligned}
p(x_1) &= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2)p(x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} p(x_1, x_2) \underbrace{\sum_{x_3 \in \{1,\ldots,K\}} p(x_3)}_{=1}
\end{aligned}
$$



Assuming independence

Let $N = 3$ and assume we can write $p(x_1, x_2, x_3) = p(x_1, x_2)p(x_3)$.

Then, we have

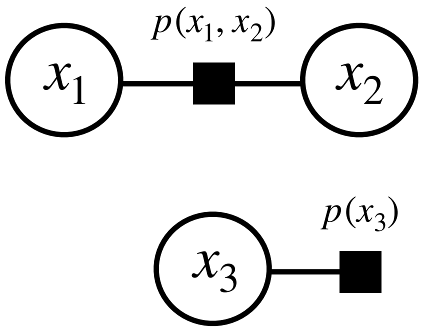


$$
\begin{aligned}
p(x_1) &= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2)p(x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} p(x_1, x_2) \underbrace{\sum_{x_3 \in \{1,\ldots,K\}} p(x_3)}_{=1} \\
&= \sum_{x_2 \in \{1,\ldots,K\}} p(x_1, x_2)
\end{aligned}
$$

Assuming independence

Let $N = 3$ and assume we can write $p(x_1, x_2, x_3) = p(x_1, x_2)p(x_3)$.

Then, we have



$$
\begin{aligned}
p(x_1) &= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2)p(x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} p(x_1, x_2) \underbrace{\sum_{x_3 \in \{1,\ldots,K\}} p(x_3)}_{=1} \\
&= \sum_{x_2 \in \{1,\ldots,K\}} p(x_1, x_2)
\end{aligned}
$$

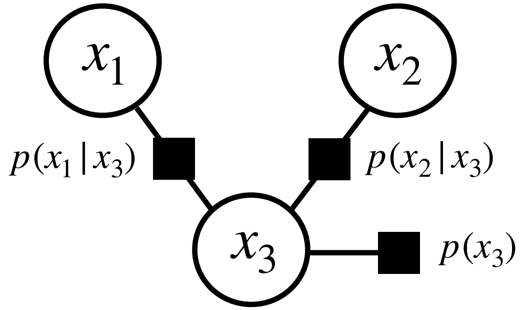
A single sum is cheaper to compute than a double sum!

Assuming conditional independence

Now assume we have $p(x_1, x_2, x_3) = p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3)$.

Assuming conditional independence

Now assume we have $p(x_1, x_2, x_3) = p(x_1 \,|\, x_3)p(x_2 \,|\, x_3)p(x_3)$.

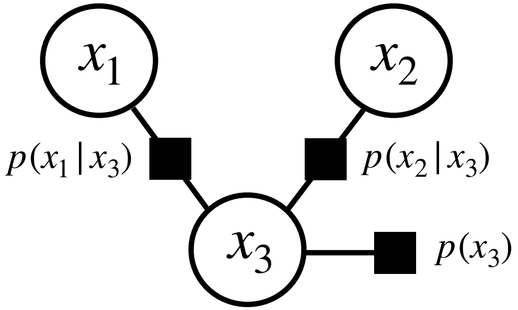


Assuming conditional independence

Now assume we have $p(x_1, x_2, x_3) = p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3)$.

Then,

$$p(x_1) = \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3)$$

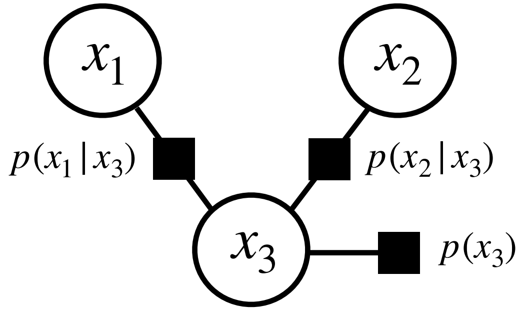


Assuming conditional independence

Now assume we have $p(x_1, x_2, x_3) = p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3)$.

Then,

$$
\begin{aligned}
p(x_1) &= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3)
\end{aligned}
$$

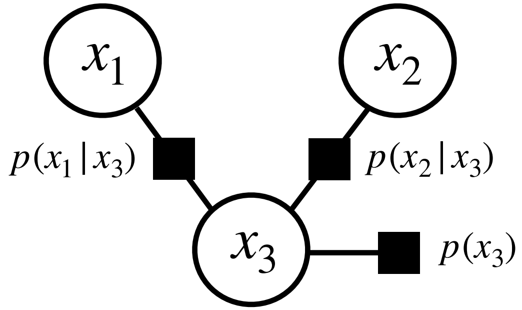


Assuming conditional independence

Now assume we have $p(x_1, x_2, x_3) = p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3)$.

Then,

$$
\begin{aligned}
p(x_1) &= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3) \\
&= \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_3) \underbrace{\sum_{x_2 \in \{1,\ldots,K\}} p(x_2 \mid x_3)}_{=1}
\end{aligned}
$$

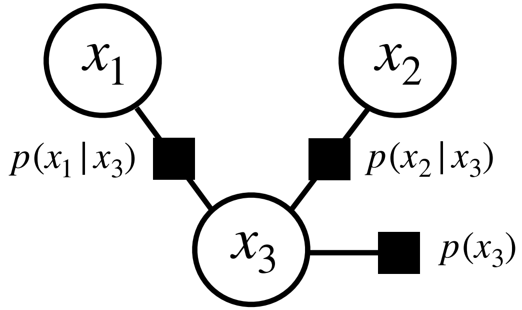


Assuming conditional independence

Now assume we have $p(x_1, x_2, x_3) = p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3)$.

Then,

$$
\begin{aligned}
p(x_1) &= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3) \\
&= \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_3) \underbrace{\sum_{x_2 \in \{1,\ldots,K\}} p(x_2 \mid x_3)}_{=1} \\
&= \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_3)
\end{aligned}
$$

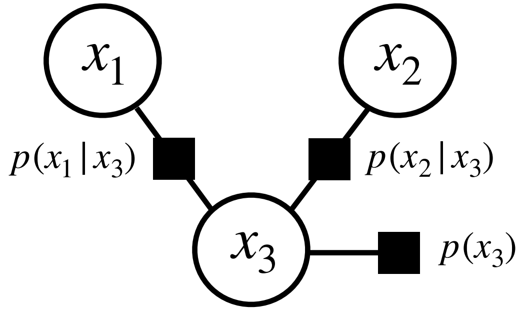


Assuming conditional independence

Now assume we have $p(x_1, x_2, x_3) = p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3)$.

Then,

$$
\begin{aligned}
p(x_1) &= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3) \\
&= \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_3) \underbrace{\sum_{x_2 \in \{1,\ldots,K\}} p(x_2 \mid x_3)}_{=1} \\
&= \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_3)
\end{aligned}
$$



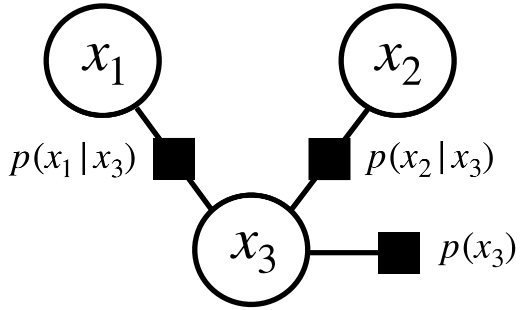
Observation: Independence / conditional independence helps to reduce complexity!

Assuming conditional independence

Now assume we have $p(x_1, x_2, x_3) = p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3)$.

Then,

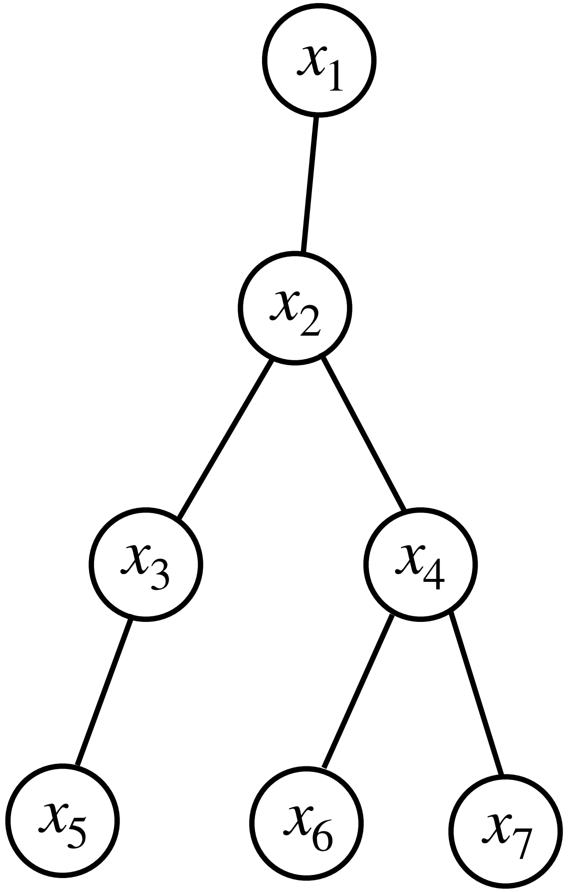
$$
\begin{aligned}
p(x_1) &= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1, x_2, x_3) \\
&= \sum_{x_2 \in \{1,\ldots,K\}} \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_2 \mid x_3)p(x_3) \\
&= \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_3) \underbrace{\sum_{x_2 \in \{1,\ldots,K\}} p(x_2 \mid x_3)}_{=1} \\
&= \sum_{x_3 \in \{1,\ldots,K\}} p(x_1 \mid x_3)p(x_3)
\end{aligned}
$$



Observation: Independence / conditional independence helps to reduce complexity!

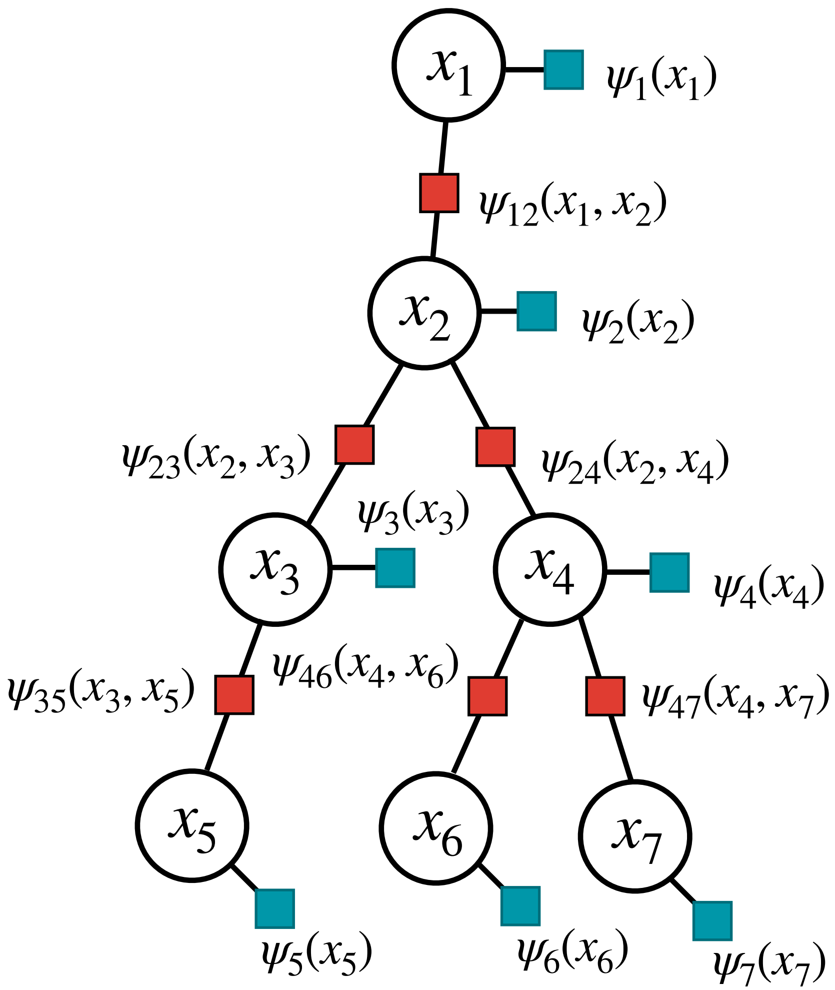
$\equiv$ *sparsity of graph*

Belief propagation algorithm



- **Belief propagation** efficiently computes *marginal probabilities* $p(x_i)$ on trees
- Assume that the graph is **tree-structured**
- Operate on factor graphs

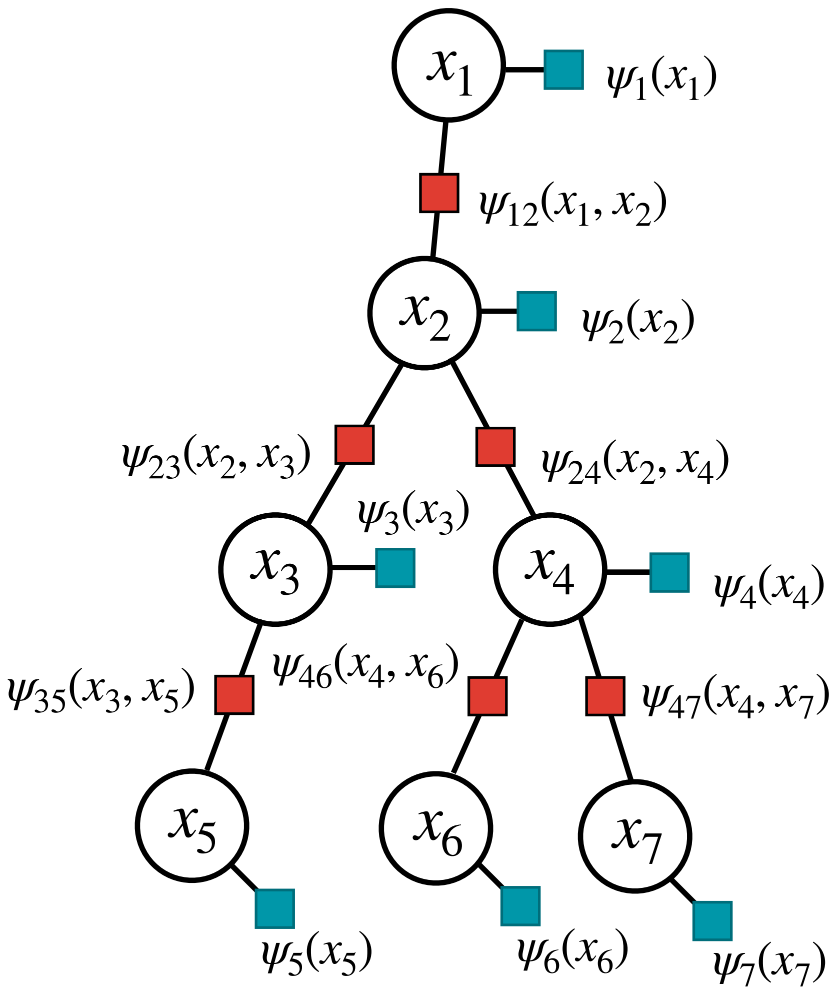
Belief propagation algorithm



- **Belief propagation** efficiently computes *marginal probabilities* $p(x_i)$ on trees
- Assume that the graph is **tree-structured**
- Operate on factor graphs

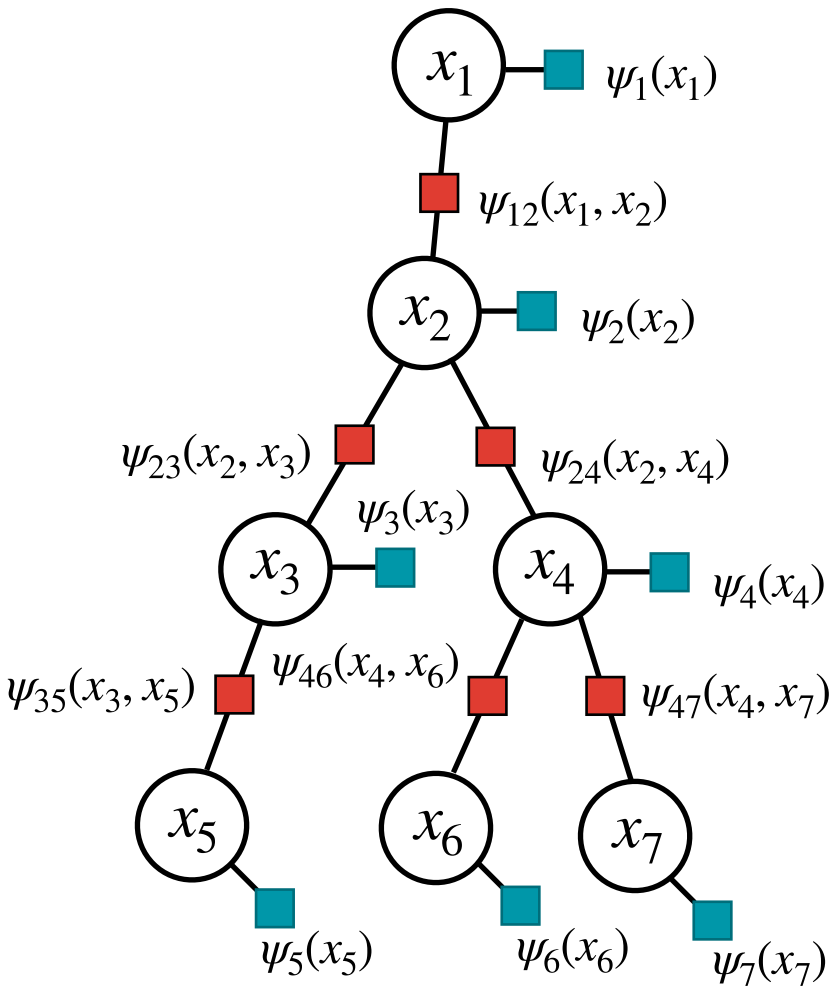
$$p(\mathbf{x}) = \prod_{i \in V} \psi_i(x_i) \prod_{(i,j) \in E} \psi_{ij}(x_i, x_j).$$

Belief propagation algorithm



BP proceeds by iteratively updating:

Belief propagation algorithm

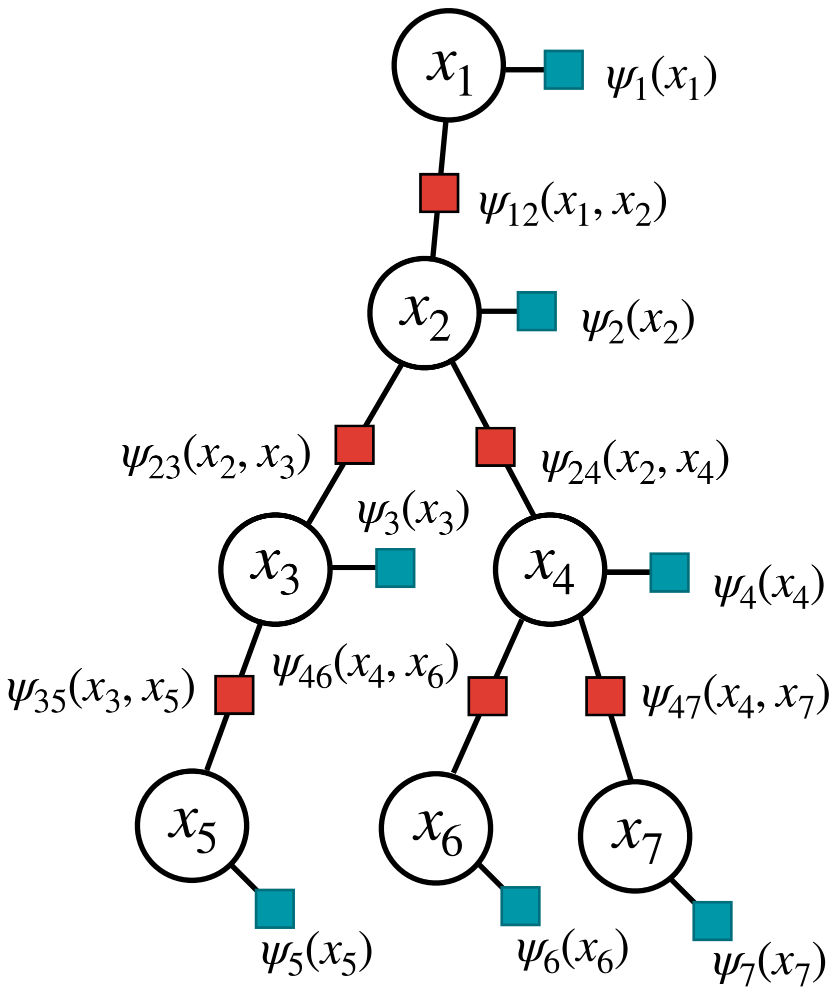


BP proceeds by iteratively updating:

1. The “**messages**” between two nodes

$$M_{j \to i}(x_i)$$

Belief propagation algorithm



BP proceeds by iteratively updating:

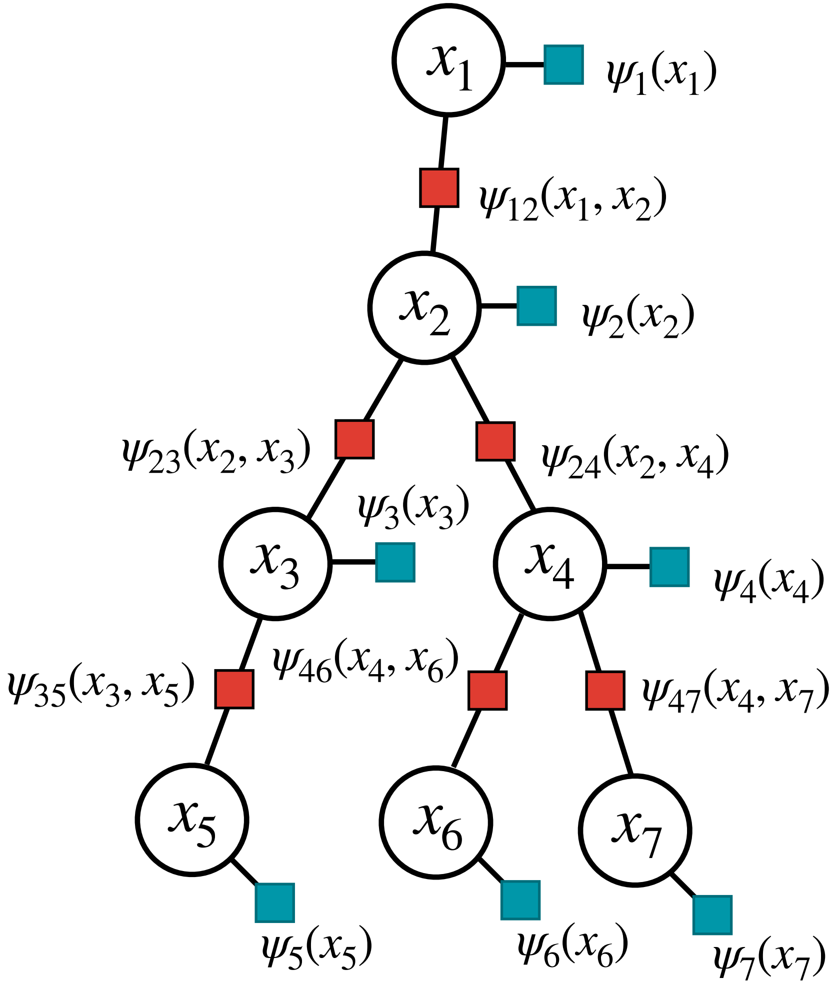
1. The "**messages**" between two nodes

$$M_{j\to i}(x_i)$$

2. The "**state**" of each node

$$p(x_i)$$

Belief propagation algorithm



BP proceeds by iteratively updating:

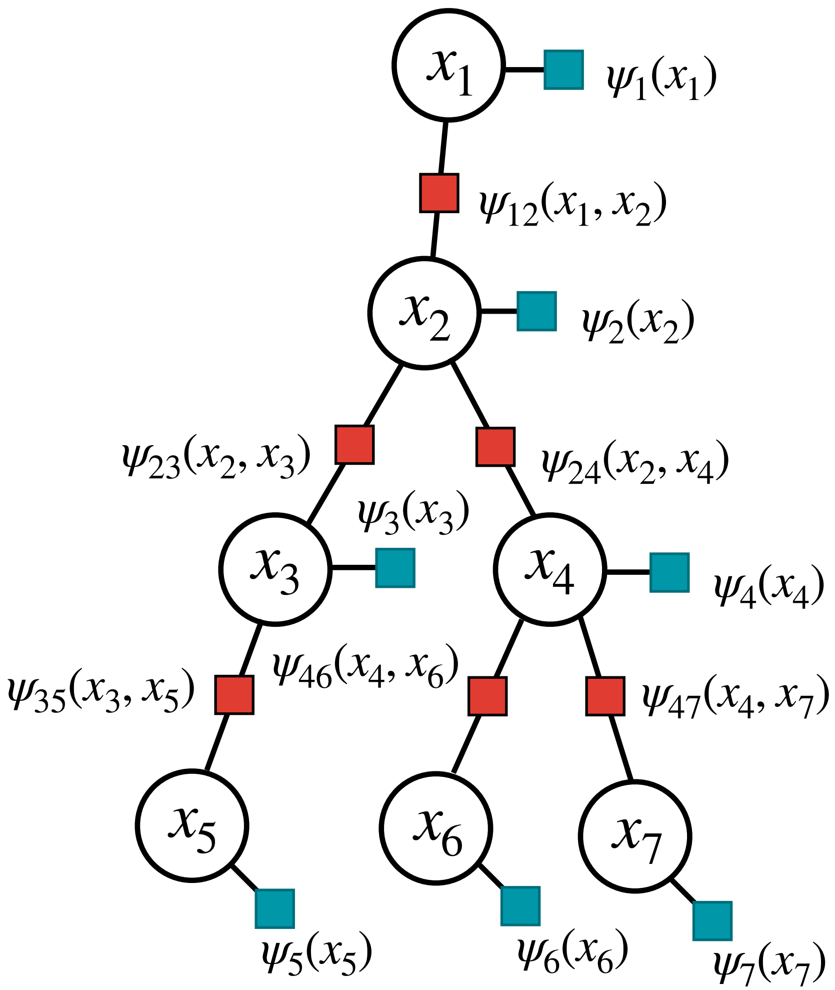
1. The “**messages**” between two nodes

$$M_{j\to i}(x_i) \to \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)$$

2. The “**state**” of each node

$$p(x_i)$$

Belief propagation algorithm



BP proceeds by iteratively updating:

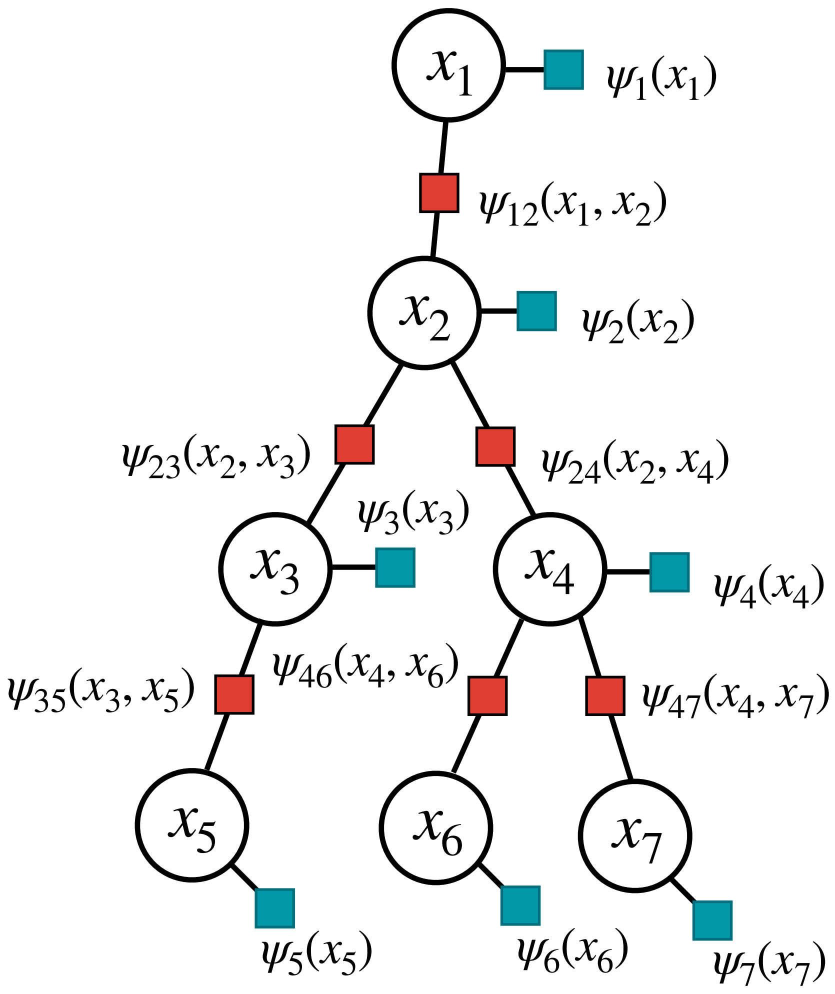
1. The “**messages**” between two nodes

$$M_{j\to i}(x_i) \to \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)$$

2. The “**state**” of each node

$$p(x_i) \to \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i)$$

Belief propagation algorithm



BP proceeds by iteratively updating:

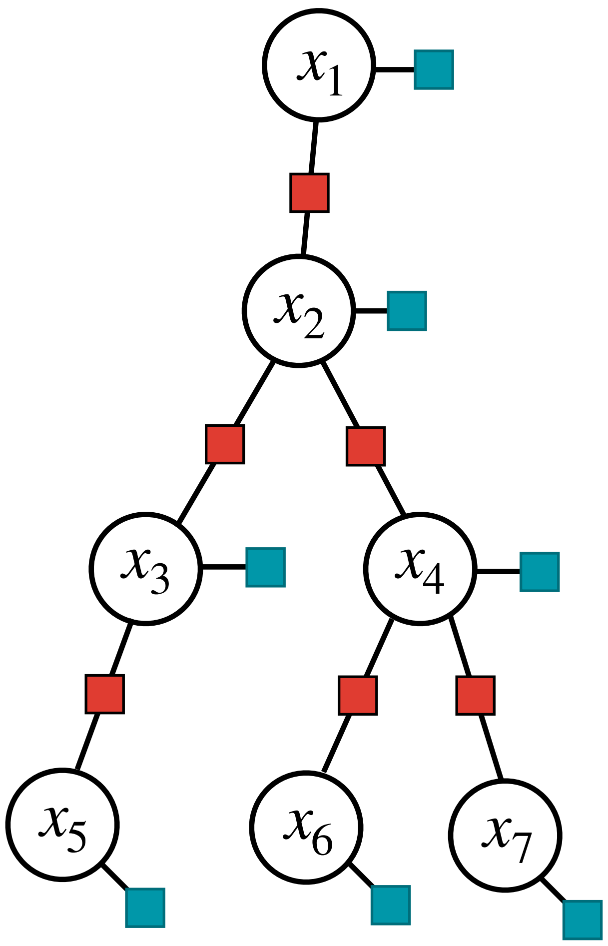
- 1. The "**messages**" between two nodes

$$M_{j\to i}(x_i) \to \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)$$

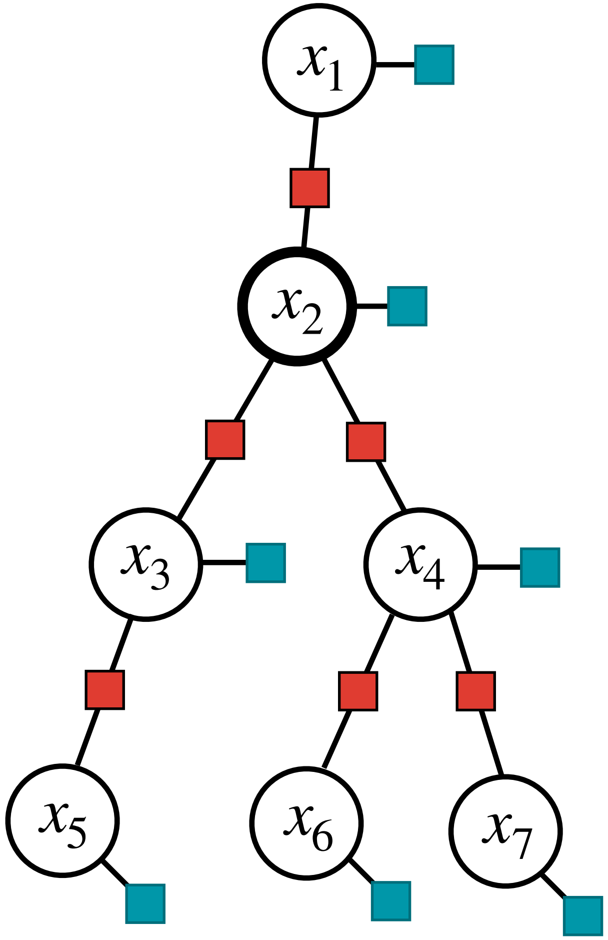
2. The "**state**" of each node

$$p(x_i) \to \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i)$$

Step 1. Message update

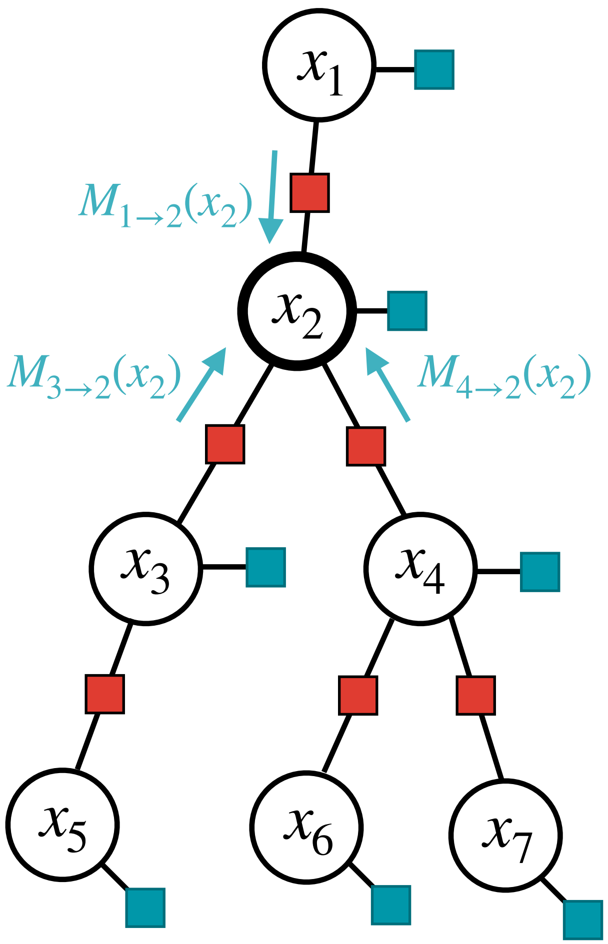


Step 1. Message update



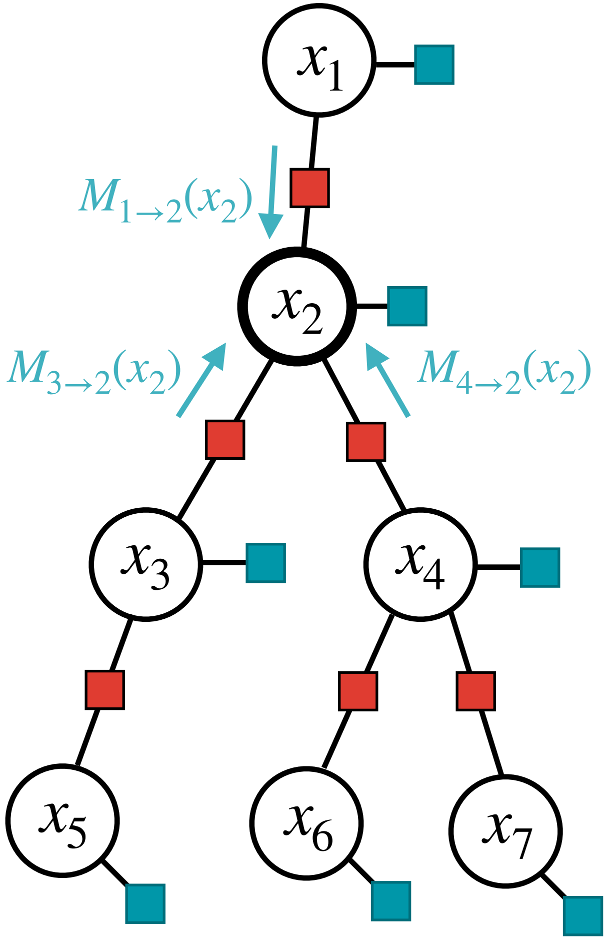
Let's say we want to compute $p(x_2)$.

Step 1. Message update



Let's say we want to compute $p(x_2)$.

Step 1. Message update

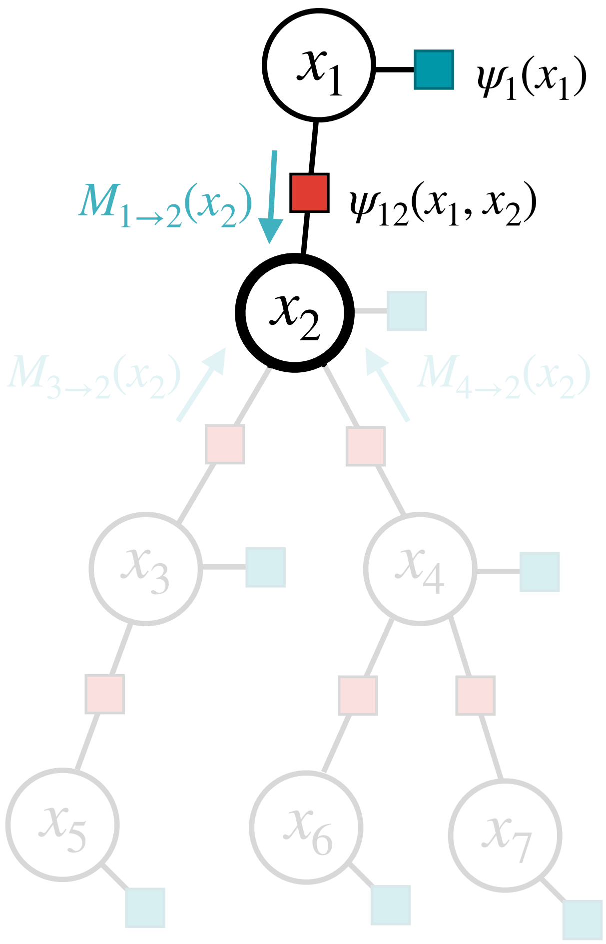


Let's say we want to compute $p(x_2)$.

Recall the message update step:

$$M_{j\to i}(x_i) = \sum_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\,\psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)$$

Step 1. Message update



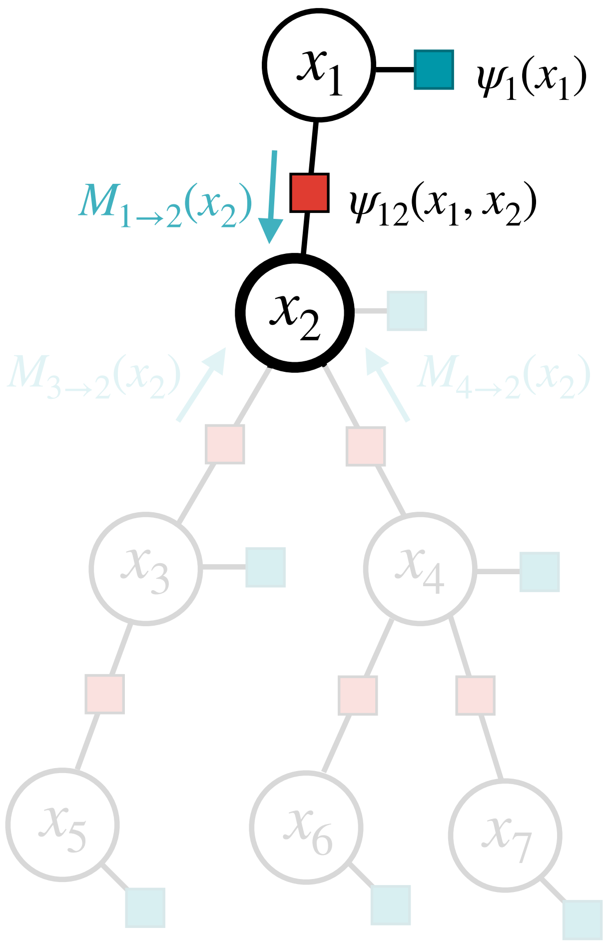
Let's say we want to compute $p(x_2)$.

Recall the message update step:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)$$

First, compute the message $x_1 \to x_2$:

Step 1. Message update



Let's say we want to compute $p(x_2)$.

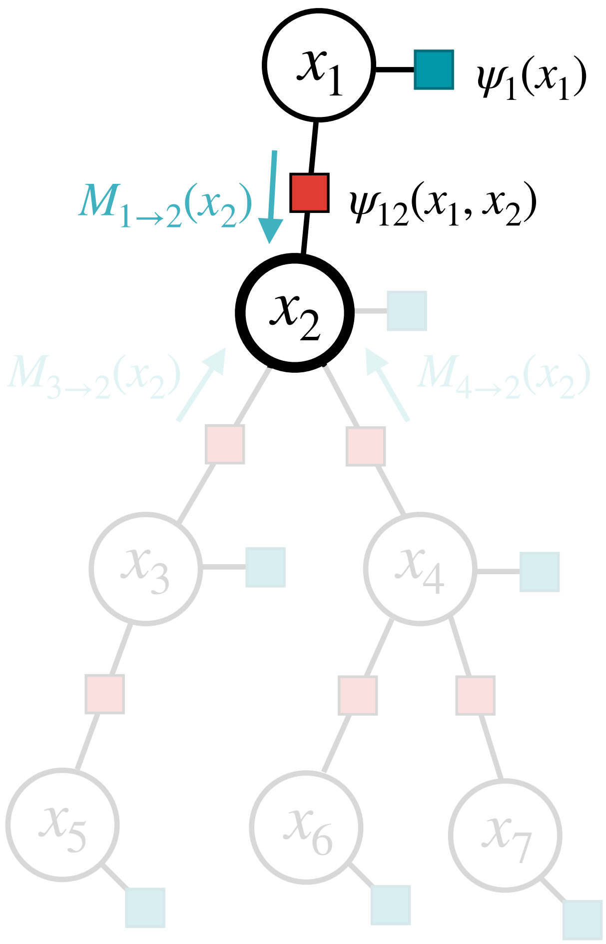
Recall the message update step:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k \sim j, k \neq i} M_{k\to j}(x_j)$$

First, compute the message $x_1 \to x_2$:

$$M_{1\to 2}(x_2) = \sum_{x_1 \in \{1,\ldots,K\}} \psi_{12}(x_1, x_2)\, \psi_1(x_1) \prod_{k \sim 1, k \neq 2} M_{k\to 1}(x_1)$$

Step 1. Message update



Let's say we want to compute $p(x_2)$.

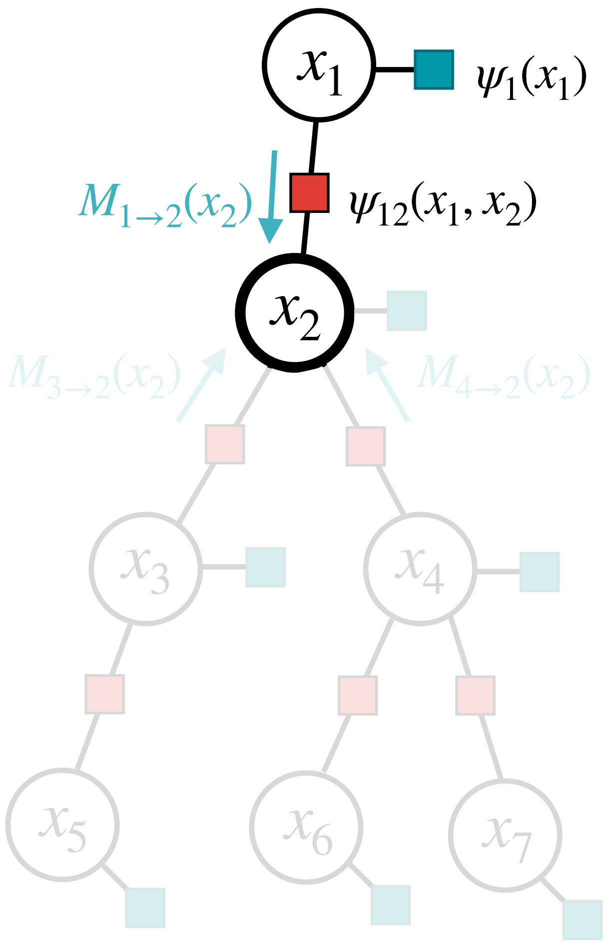
Recall the message update step:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)$$

First, compute the message $x_1 \to x_2$:

$$M_{1\to2}(x_2) = \sum_{x_1 \in \{1,\ldots,K\}} \psi_{12}(x_1, x_2)\, \psi_1(x_1) \underbrace{\prod_{k\sim 1, k\neq 2} M_{k\to 1}(x_1)}_{??}$$

Step 1. Message update



Let's say we want to compute $p(x_2)$.

Recall the message update step:

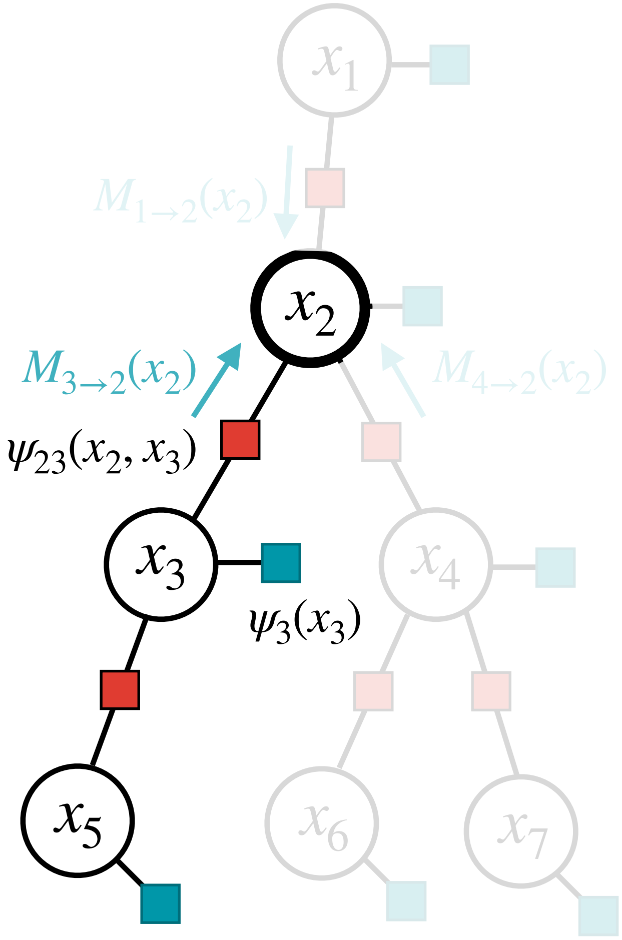
$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)$$

First, compute the message $x_1 \to x_2$:

$$\boxed{M_{1\to 2}(x_2)} = \sum_{x_1 \in \{1,\ldots,K\}} \psi_{12}(x_1, x_2)\, \psi_1(x_1) \cancel{\prod_{k\sim 1, k\neq 2} M_{k\to 1}(x_1)}$$

Rule: Ignore "incoming messages" to node i if there are none

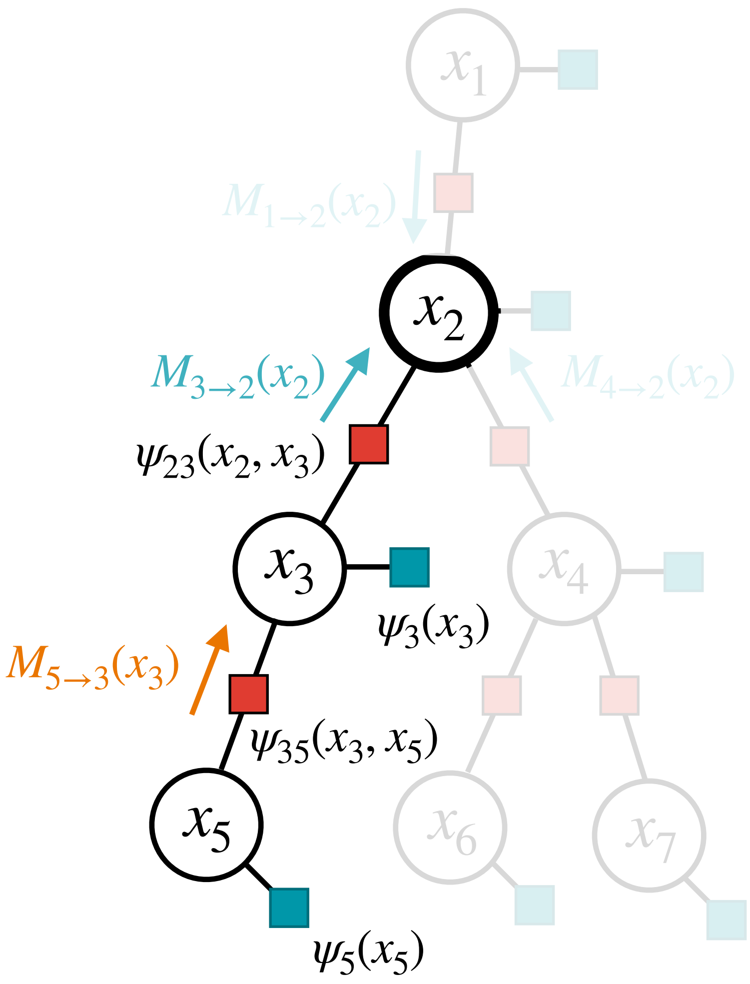
Step 1. Message update



Next, compute the message $x_3 \to x_2$:

$$\boxed{M_{3\to 2}(x_2)} = \sum_{x_3 \in \{1,\ldots,K\}} \psi_{23}(x_2, x_3)\, \psi_3(x_3) \underbrace{\prod_{k \sim 3, k \neq 2} M_{k\to 3}(x_3)}_{??}$$

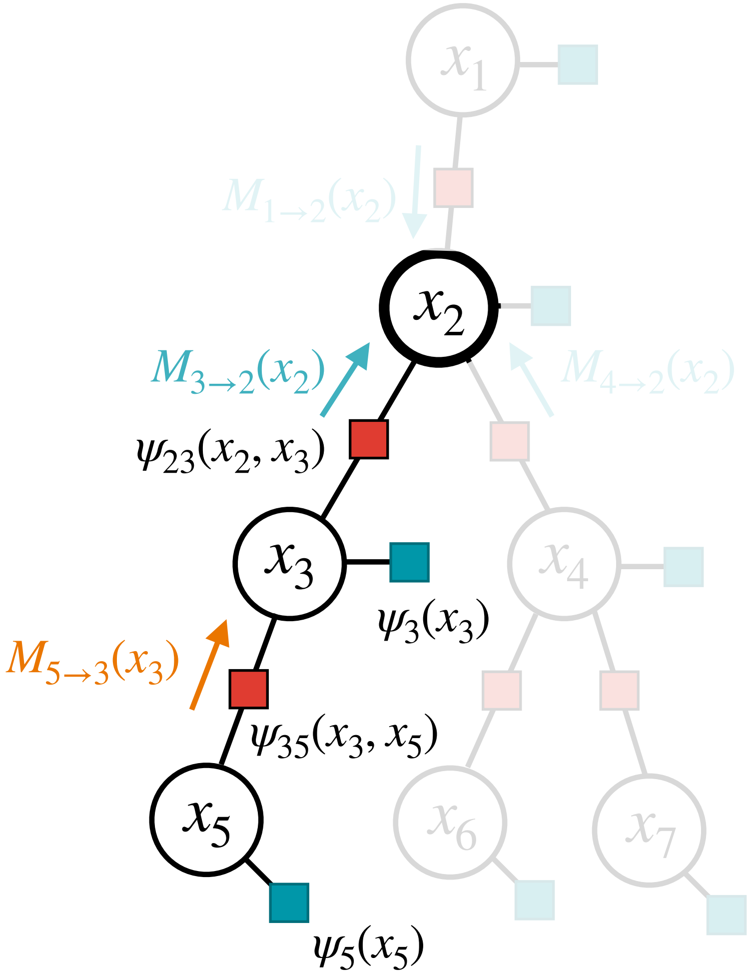
Step 1. Message update



Next, compute the message $x_3 \to x_2$:

$$M_{3\to 2}(x_2) = \sum_{x_3 \in \{1,\ldots,K\}} \psi_{23}(x_2, x_3)\, \psi_3(x_3) M_{5\to 3}(x_3)$$

Step 1. Message update

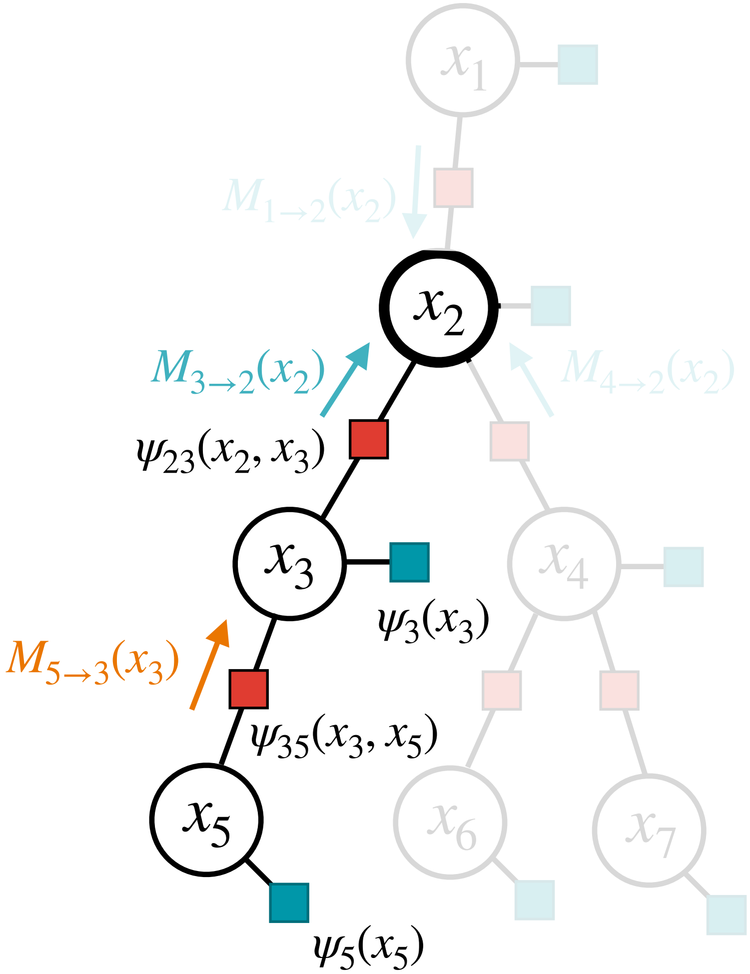


Next, compute the message $x_3 \to x_2$:

$$M_{3\to 2}(x_2) = \sum_{x_3 \in \{1,\ldots,K\}} \psi_{23}(x_2, x_3)\, \psi_3(x_3) M_{5\to 3}(x_3)$$

$$M_{5\to 3}(x_3) = \sum_{x_5 \in \{1,\ldots,K\}} \psi_{35}(x_3, x_5)\psi_5(x_5) \prod_{k\sim 5, k\neq 3} M_{k\to 5}(x_5)$$

Step 1. Message update

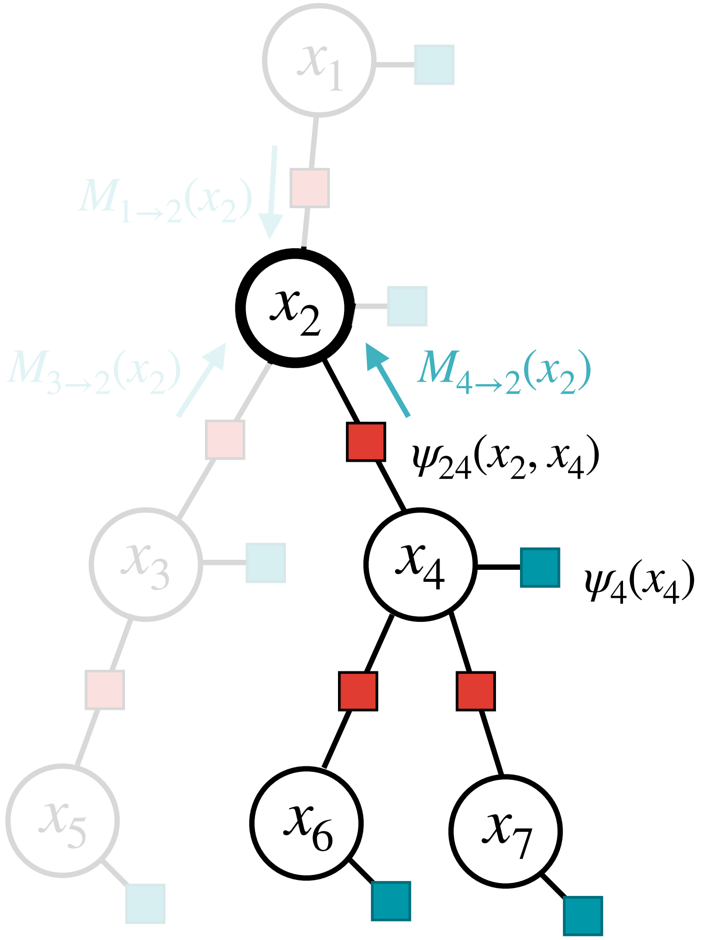


Next, compute the message $x_3 \to x_2$:

$$M_{3\to 2}(x_2) = \sum_{x_3 \in \{1,\ldots,K\}} \psi_{23}(x_2, x_3)\, \psi_3(x_3) M_{5\to 3}(x_3)$$

$$M_{5\to 3}(x_3) = \sum_{x_5 \in \{1,\ldots,K\}} \psi_{35}(x_3, x_5)\psi_5(x_5) \cancel{\prod_{k\sim 5, k\neq 3} M_{k\to 5}(x_5)}$$

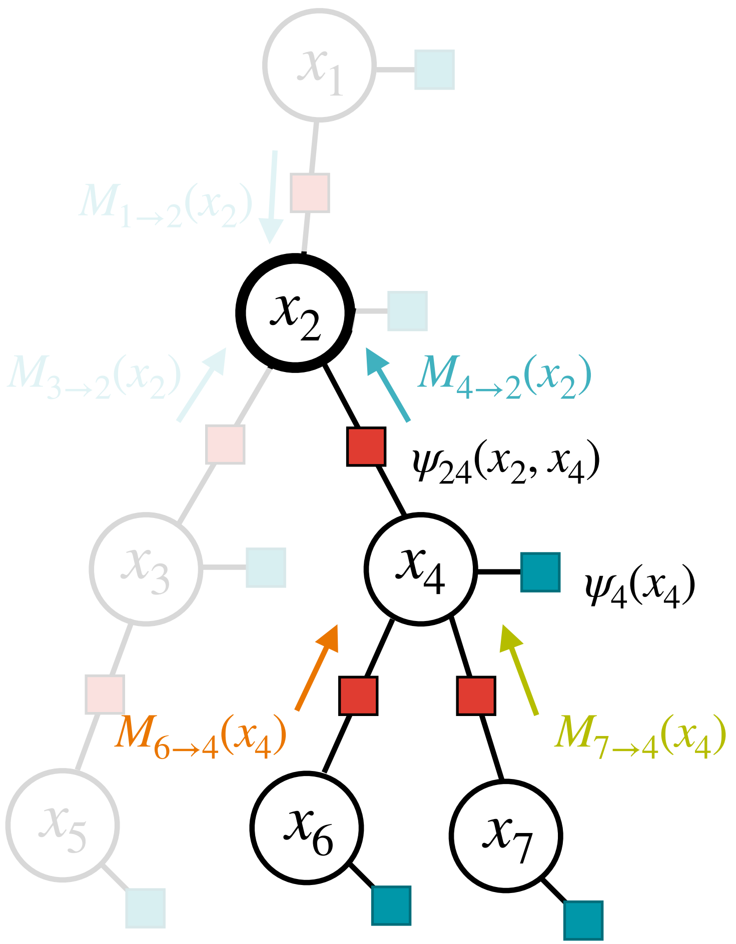
Step 1. Message update



Finally, compute the message $x_4 \to x_2$:

$$\boxed{M_{4\to2}(x_2)} = \sum_{x_4\in\{1,\ldots,K\}} \psi_{24}(x_2, x_4)\, \psi_4(x_4) \underbrace{\prod_{k\sim4,k\neq2} M_{k\to4}(x_4)}_{??}$$

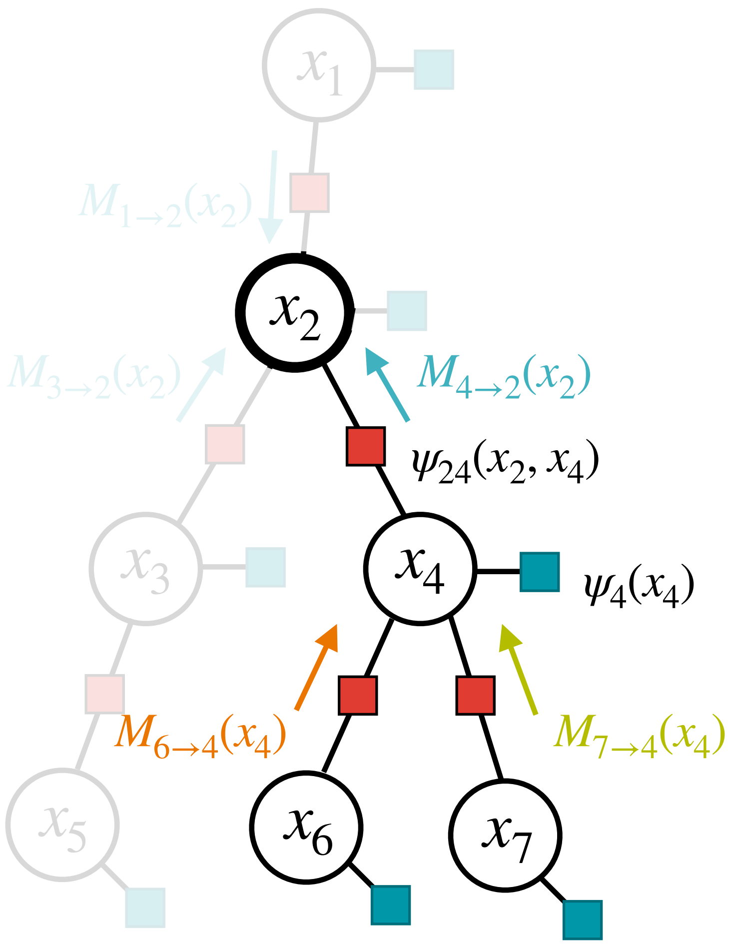
Step 1. Message update



Finally, compute the message $x_4 \to x_2$:

$$M_{4\to2}(x_2) = \sum_{x_4\in\{1,\ldots,K\}} \psi_{24}(x_2, x_4)\, \psi_4(x_4)\, M_{6\to4}(x_4)\, M_{7\to4}(x_4)$$

Step 1. Message update



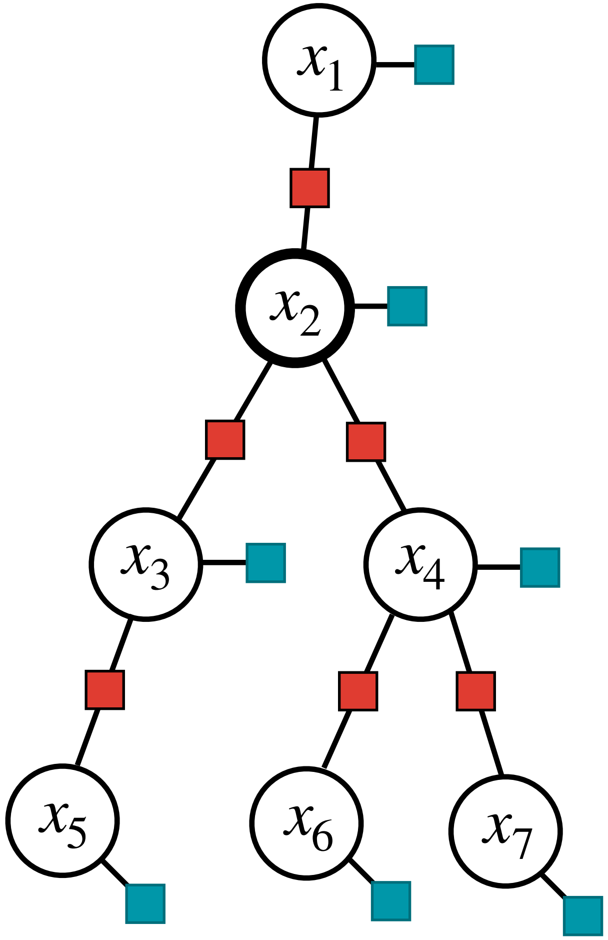
Finally, compute the message $x_4 \to x_2$:

$$M_{4\to 2}(x_2) = \sum_{x_4 \in \{1,\ldots,K\}} \psi_{24}(x_2, x_4)\, \psi_4(x_4)\, M_{6\to 4}(x_4)\, M_{7\to 4}(x_4)$$

$$M_{6\to 4}(x_4) = \sum_{x_6 \in \{1,\ldots,K\}} \psi_{46}(x_4, x_6)\psi_6(x_6)$$

$$M_{7\to 4}(x_4) = \sum_{x_7 \in \{1,\ldots,K\}} \psi_{47}(x_4, x_7)\psi_7(x_7)$$

Belief propagation algorithm



BP proceeds by iteratively updating:

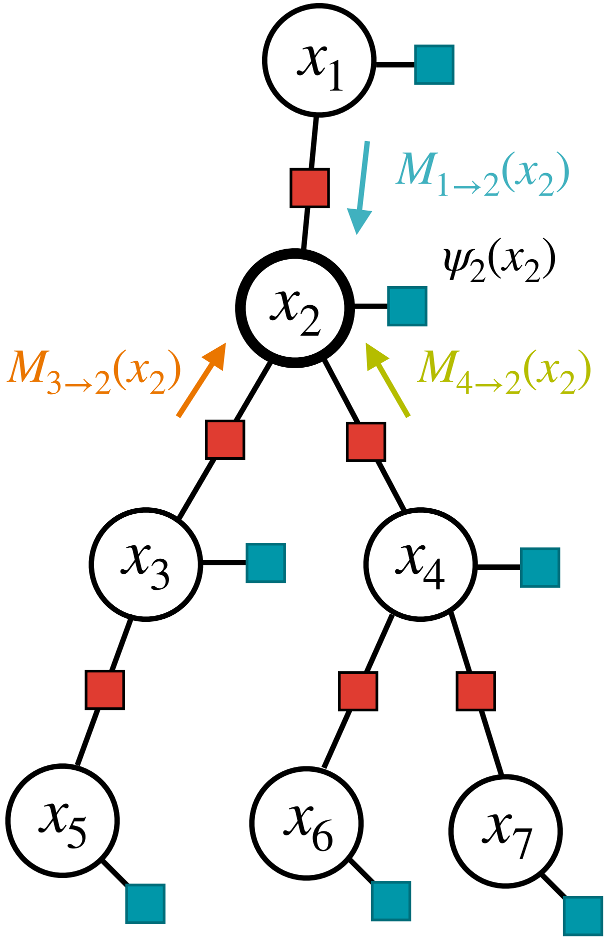
1. The "**messages**" between two nodes

$$M_{j\to i}(x_i) \to \sum_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\,\psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)$$

- 2. The "**state**" of each node

$$p(x_i) \to \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i)$$

Step 2. State update



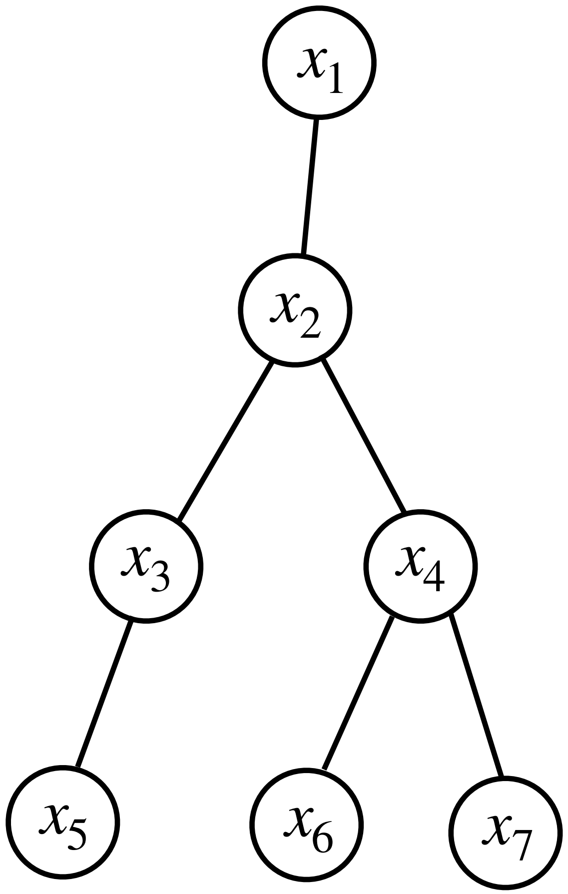
Now we can compute $p(x_2)$:

$$p(x_2) = \frac{1}{Z}\,\psi_2(x_2) \times M_{1\to2}(x_2) \times M_{3\to2}(x_2) \times M_{4\to2}(x_2)$$

where

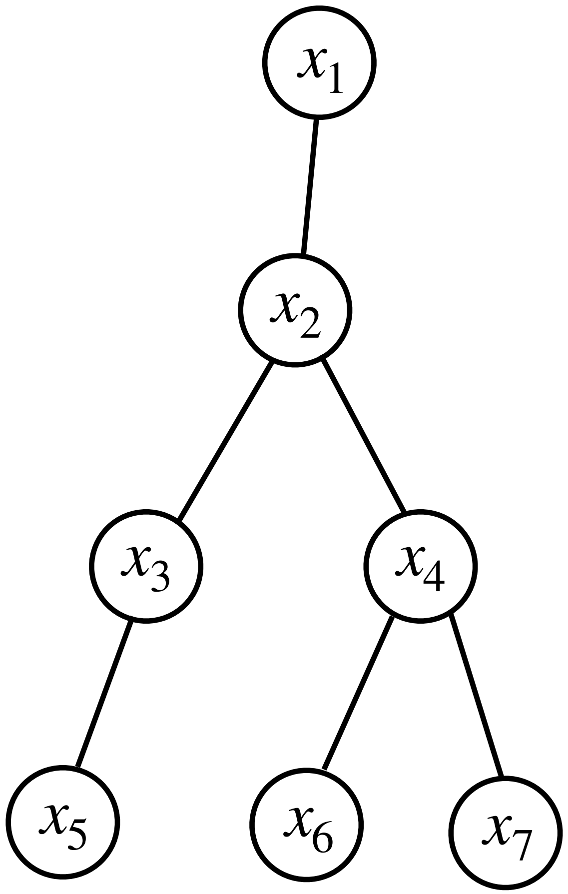
$$Z = \sum_{x_2 \in \{1,\ldots,K\}} \psi_2(x_2) \times M_{1\to2}(x_2) \times M_{3\to2}(x_2) \times M_{4\to2}(x_2)$$

Efficient implementation



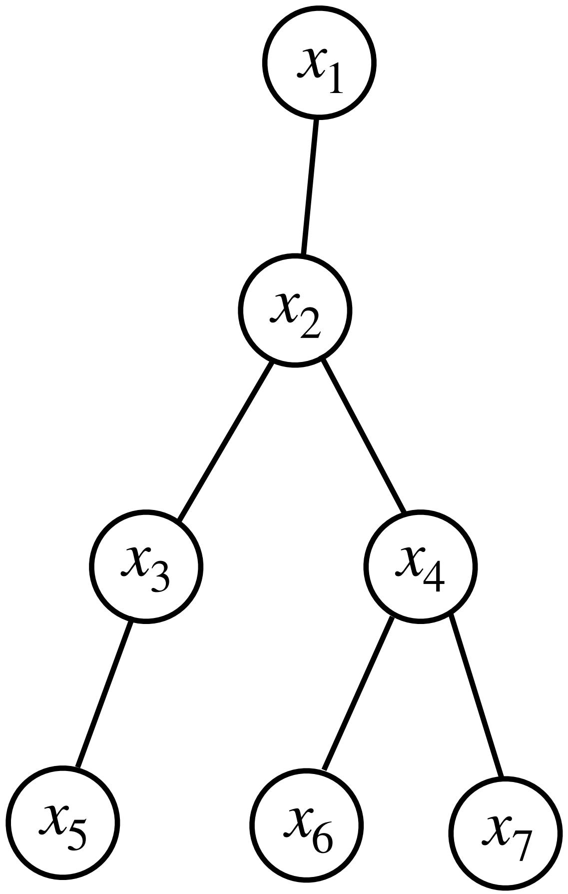
Efficient implementation

Exploiting the tree-structure, we can compute *all* the marginals efficiently



Efficient implementation

Step 0. Initialise



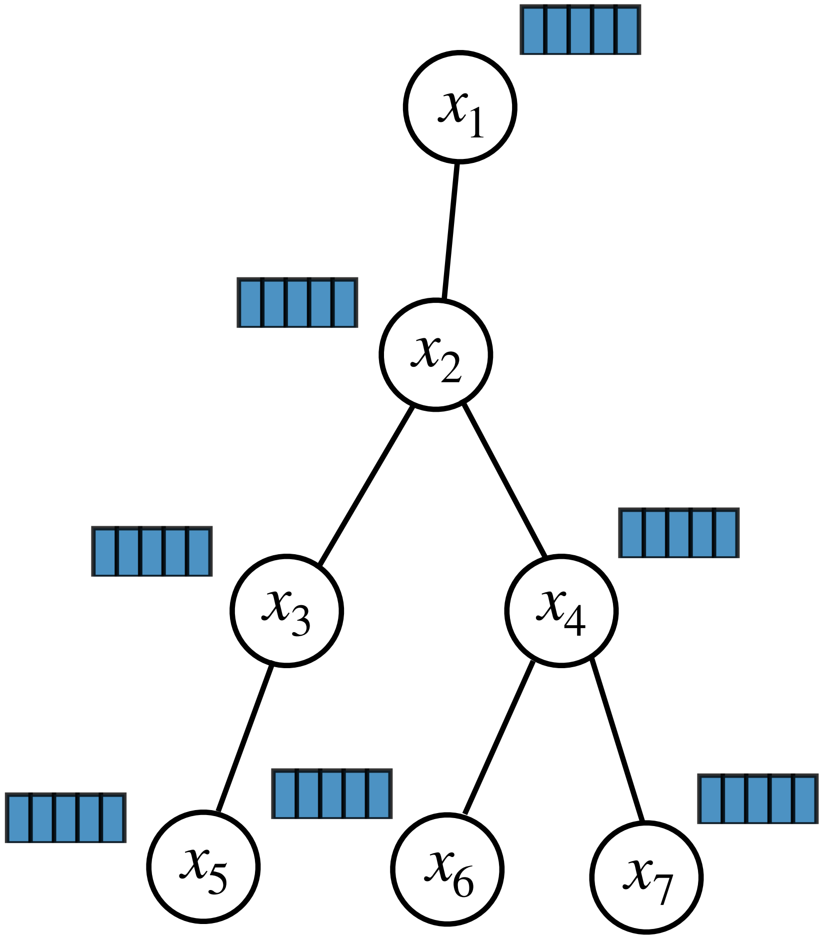
Efficient implementation

Step 0. Initialise

- the **states** as

$$p(x_i) = \frac{1}{K}\mathbf{1},$$

for all $i \in V$, and



Efficient implementation

Step 0. Initialise

- the **states** as

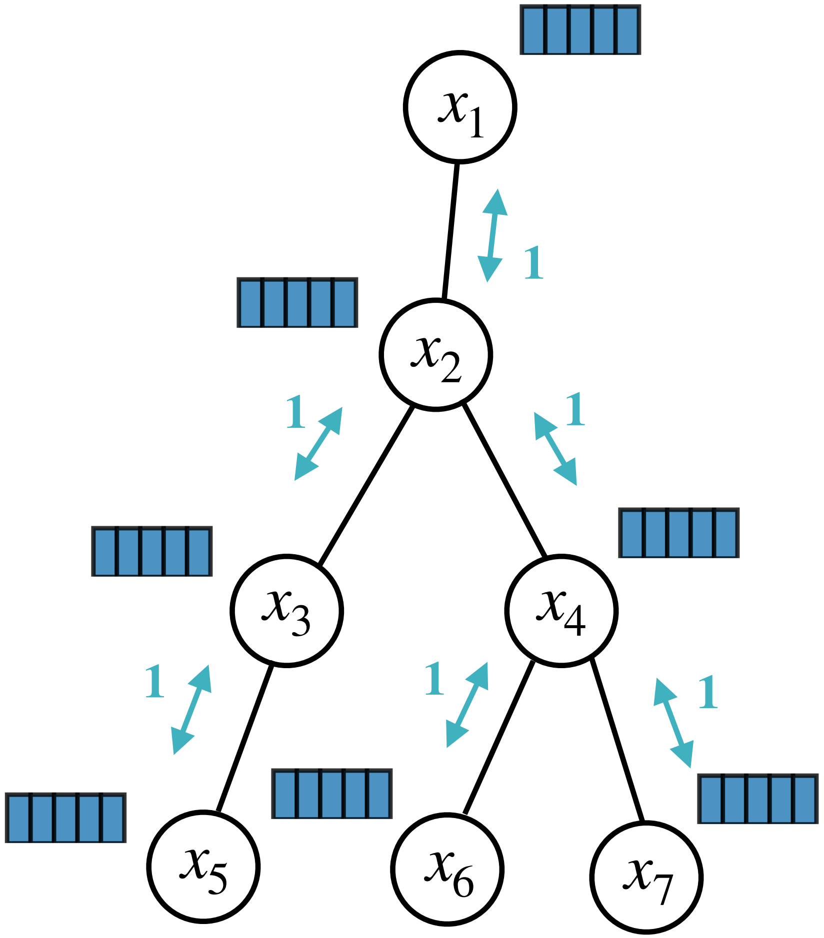
$$p(x_i) = \frac{1}{K}\mathbf{1},$$

  for all $i \in V$, and

- the **messages** as

$$M_{j\to i}(x_i) = \mathbf{1}$$

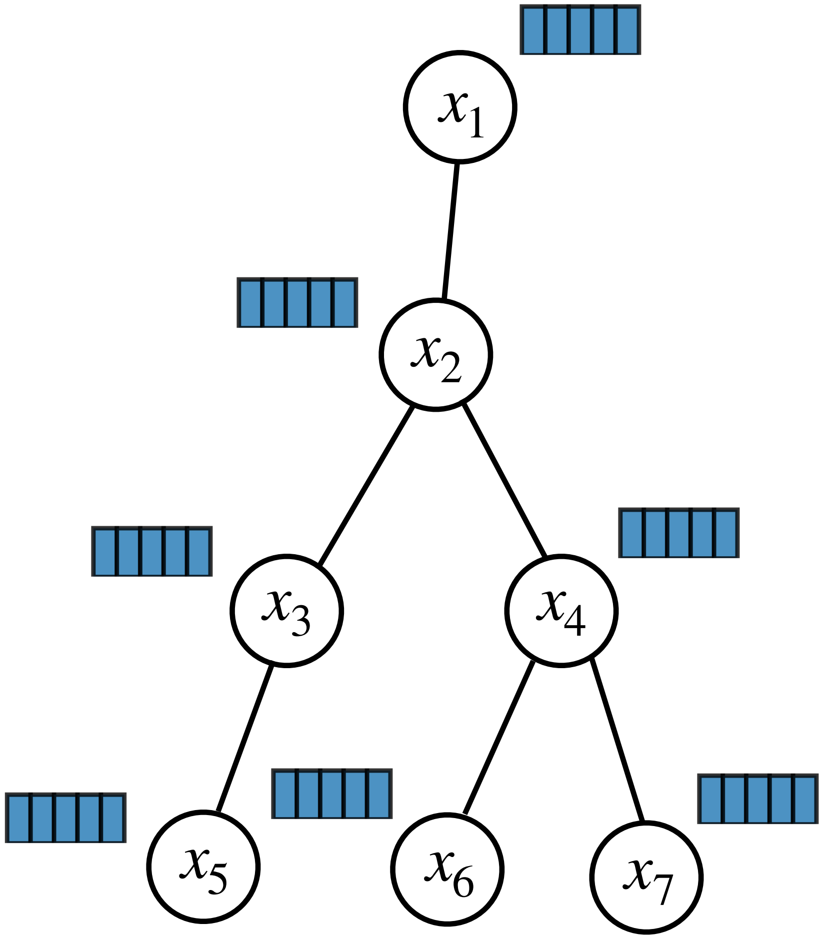
  for all $(i, j) \in E$.



Efficient implementation

Step 1. Choose a "**root**" node and identify the corresponding "**leaf**" nodes

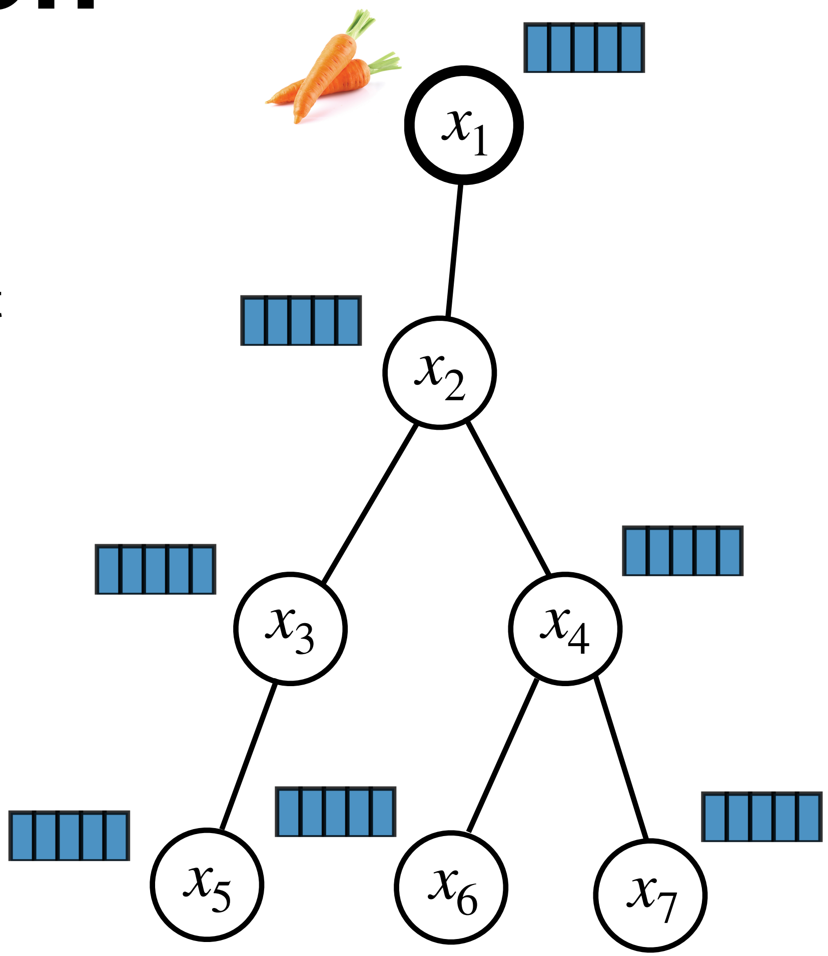
Note: The leaves are the furthest descendants of the root



Efficient implementation

Step 1. Choose a "**root**" node and identify the corresponding "**leaf**" nodes

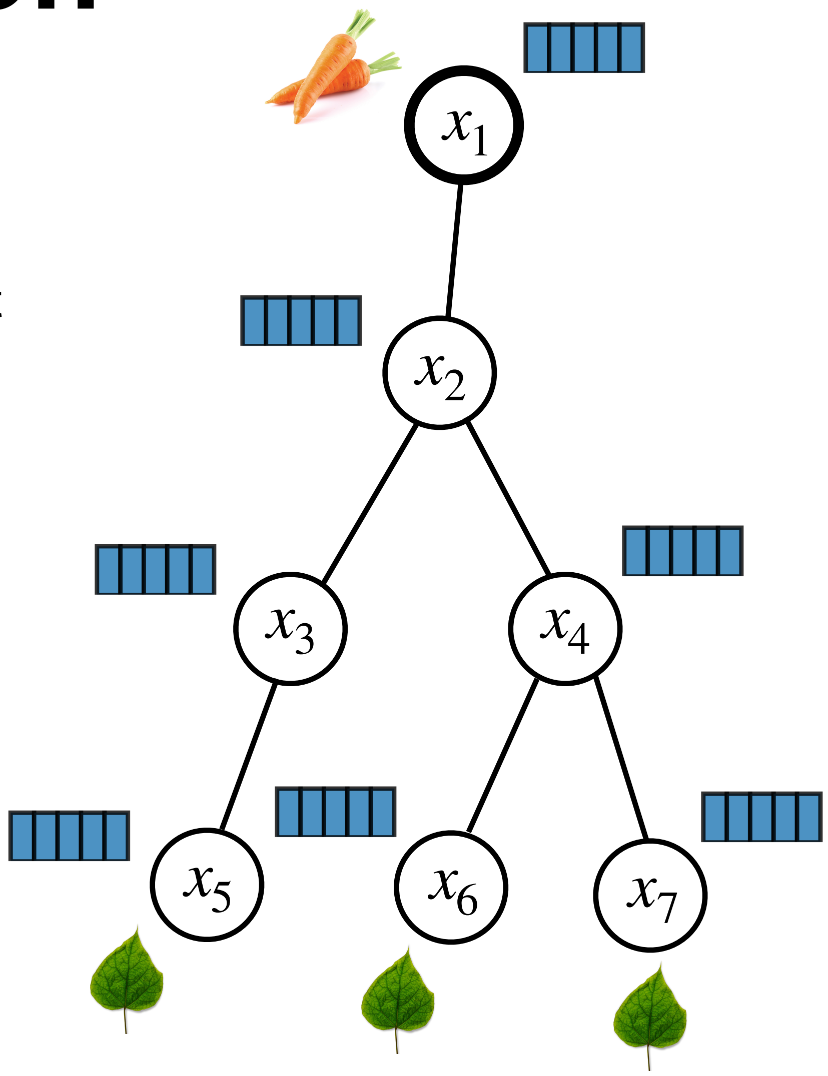
Note: The leaves are the furthest descendants of the root



Efficient implementation

Step 1. Choose a “**root**” node and identify the corresponding “**leaf**” nodes

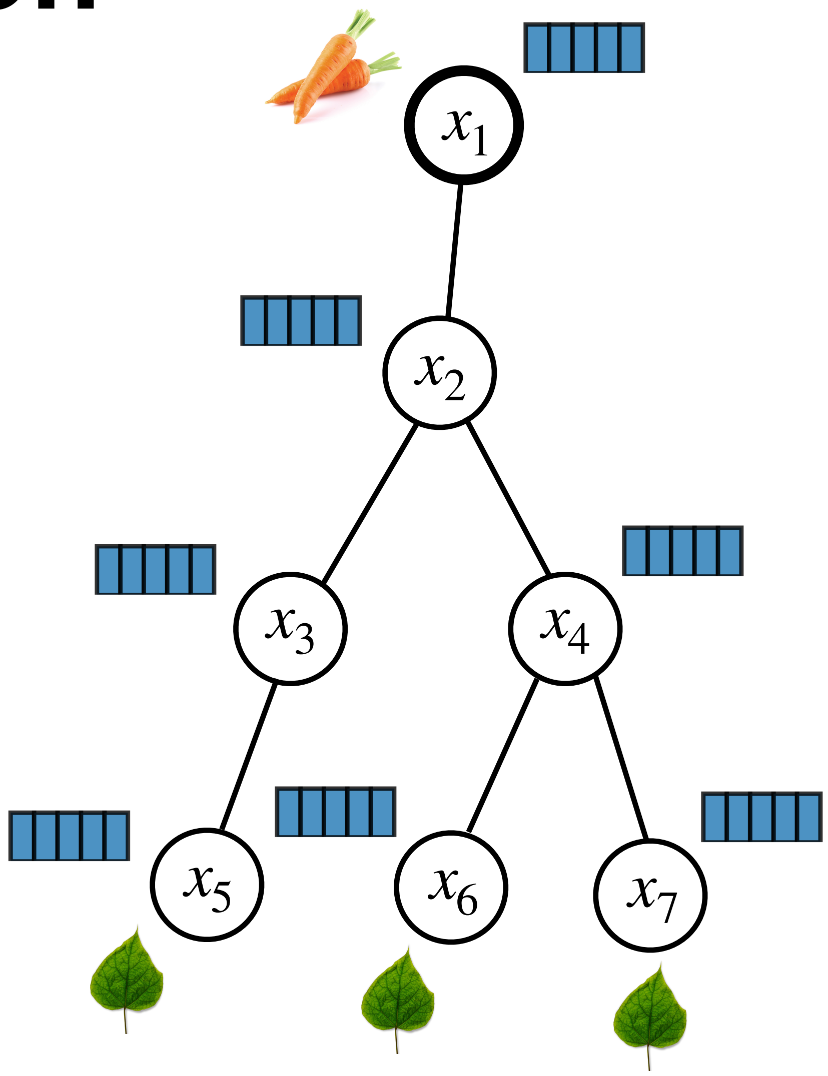
Note: The leaves are the furthest descendants of the root



Efficient implementation

Step 2. Update

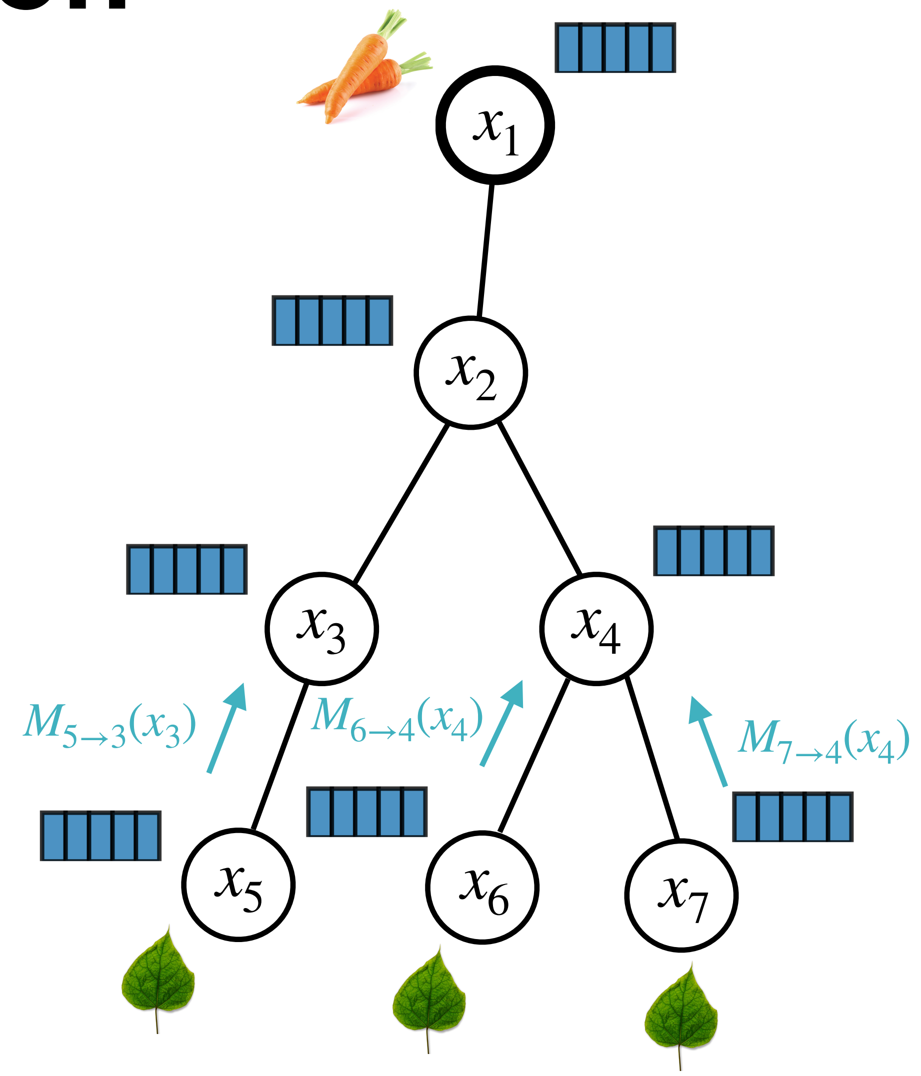
- all **messages** propagating from the leaf nodes, and
- all the **states** of their parent nodes



Efficient implementation

Step 2. Update

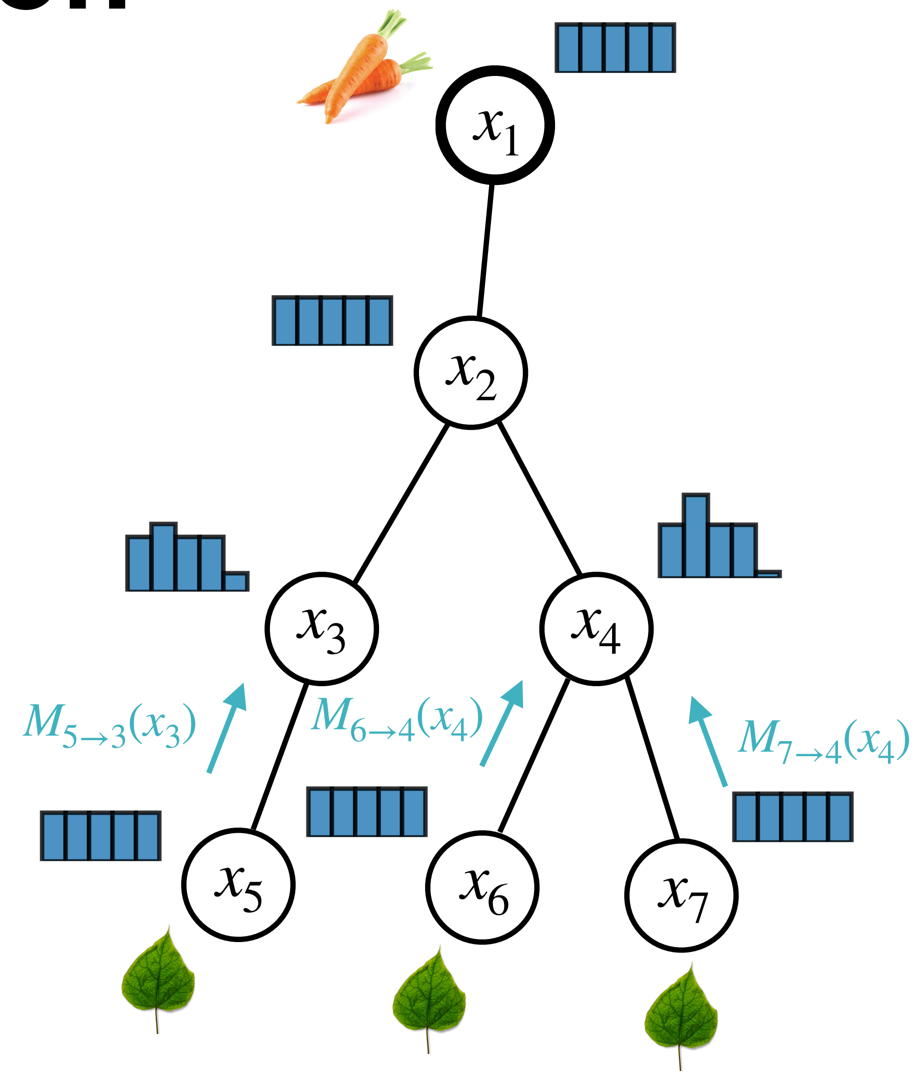
- all **messages** propagating from the leaf nodes, and
- all the **states** of their parent nodes



Efficient implementation

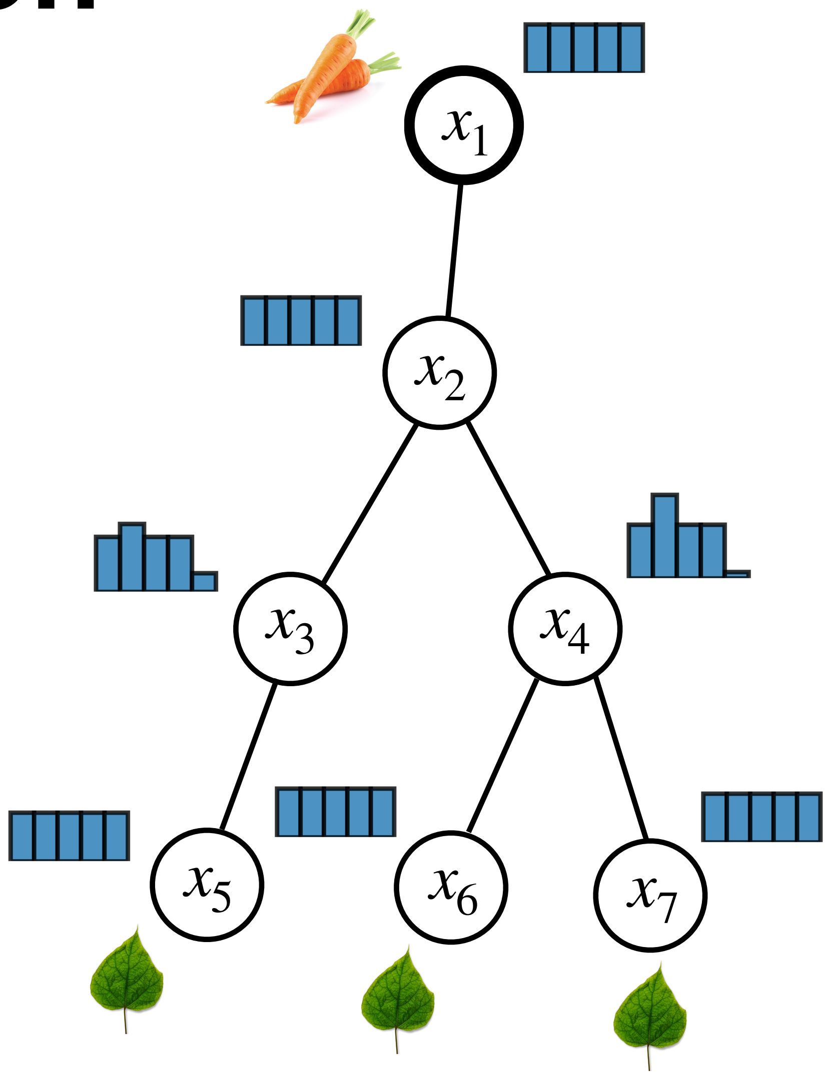
Step 2. Update

- all **messages** propagating from the leaf nodes, and
- all the **states** of their parent nodes



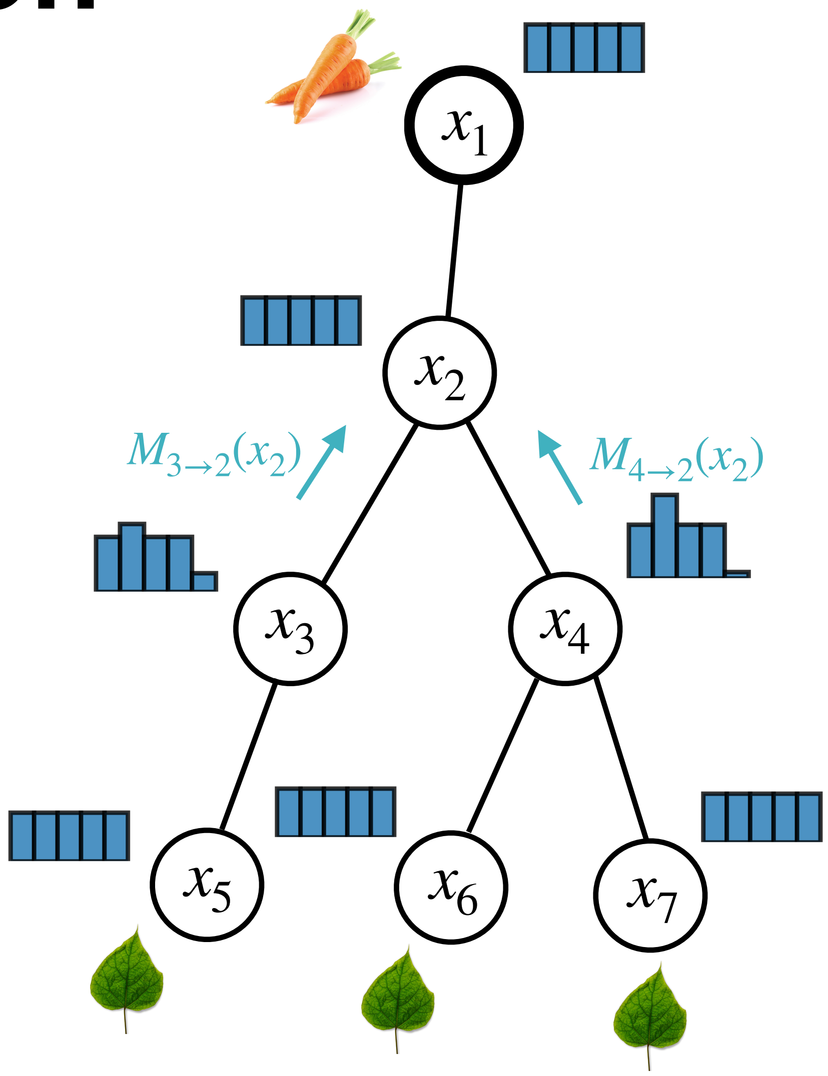
Efficient implementation

Step 3. Update the **messages** and **states** all the way up to the root



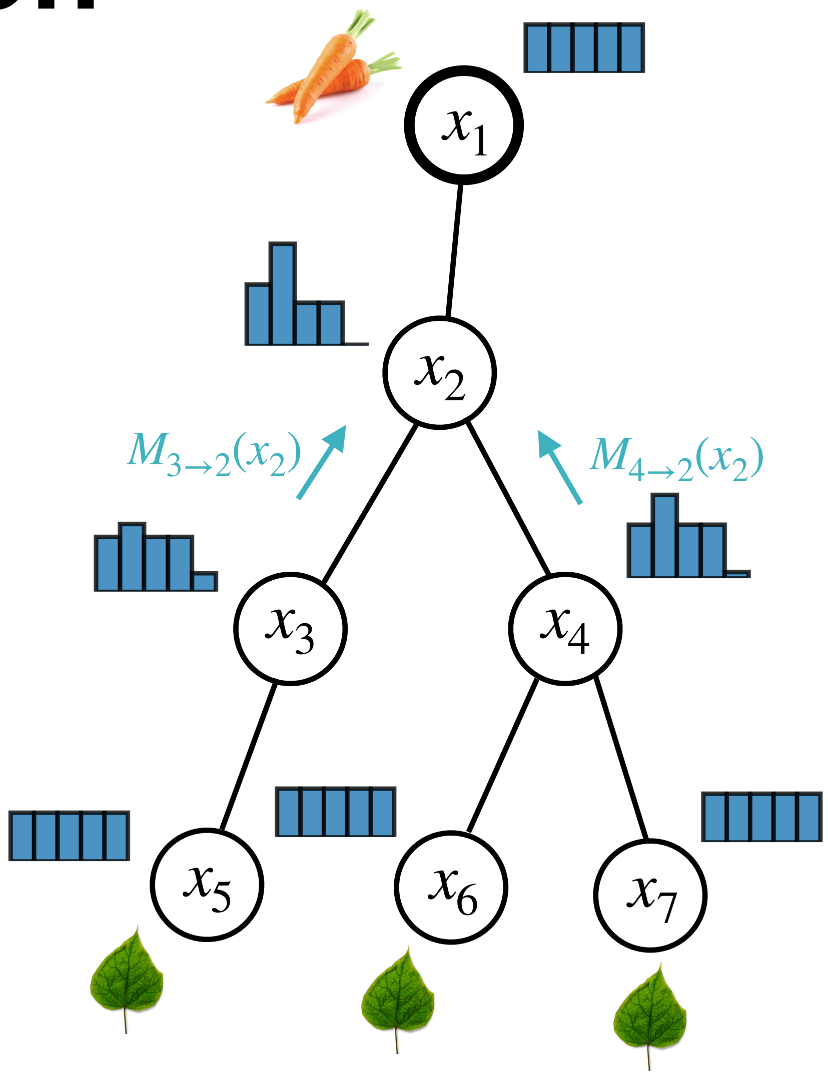
Efficient implementation

Step 3. Update the **messages** and **states** all the way up to the root



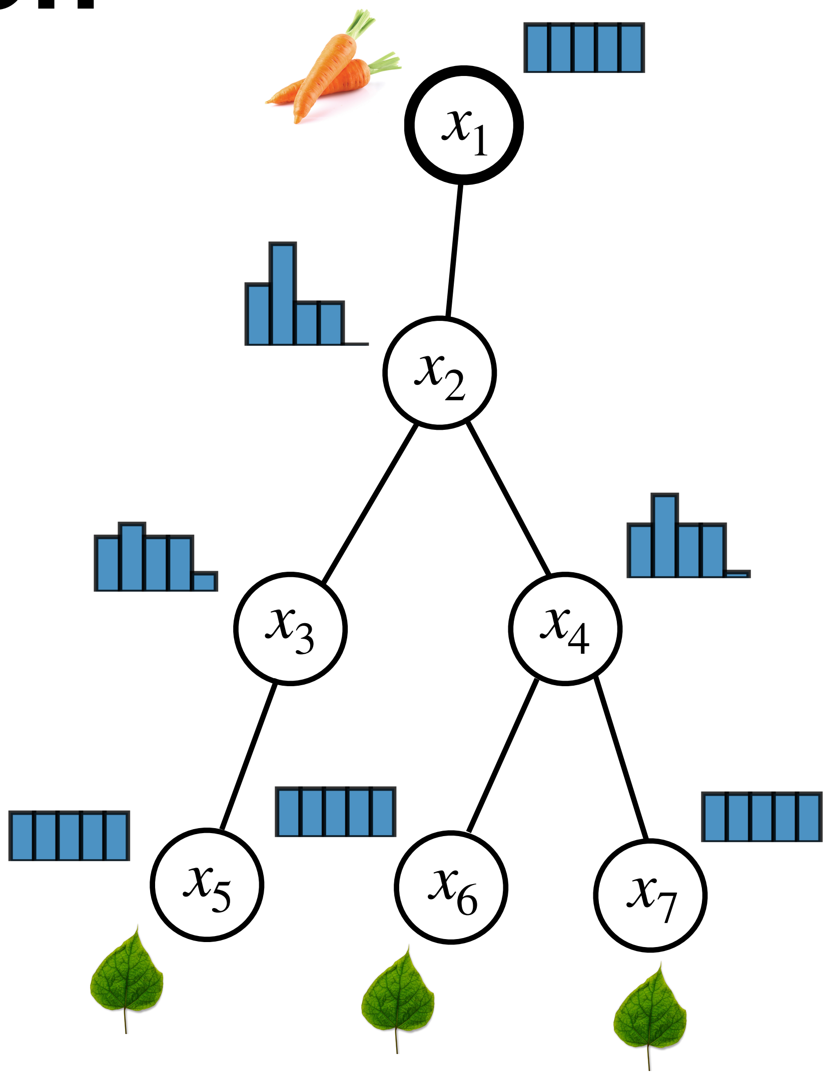
Efficient implementation

Step 3. Update the **messages** and **states** all the way up to the root



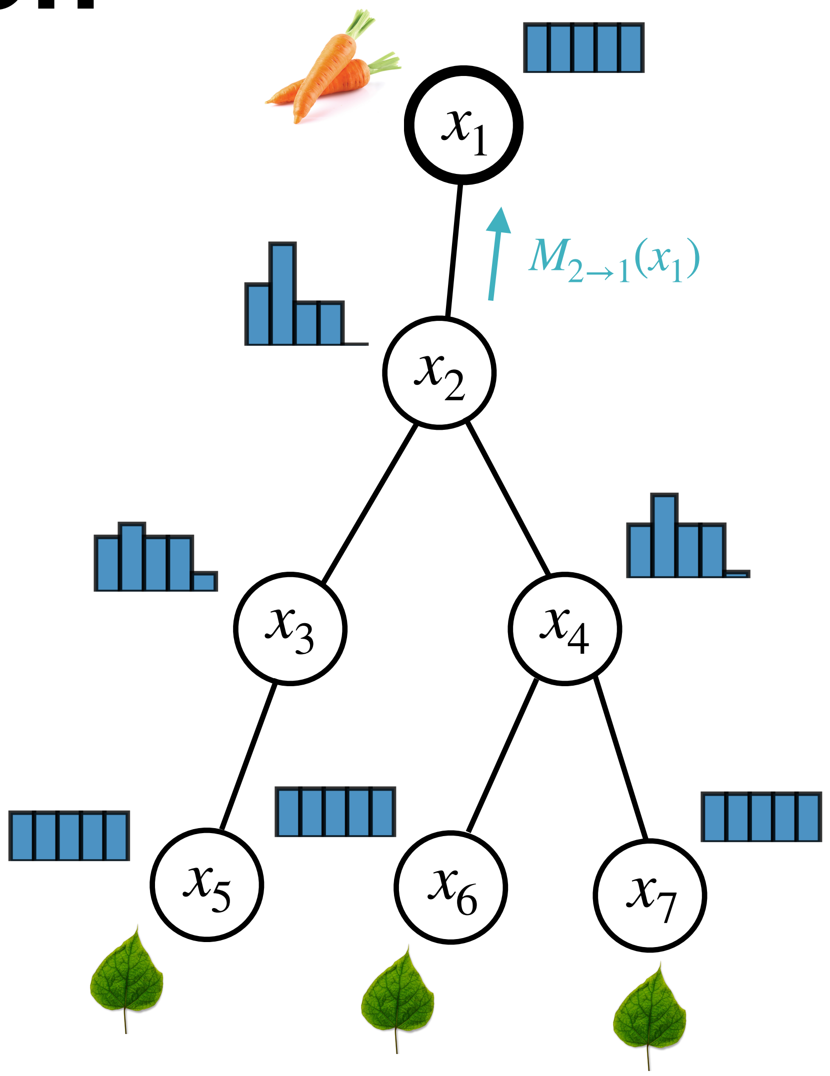
Efficient implementation

Step 3. Update the **messages** and **states** all the way up to the root



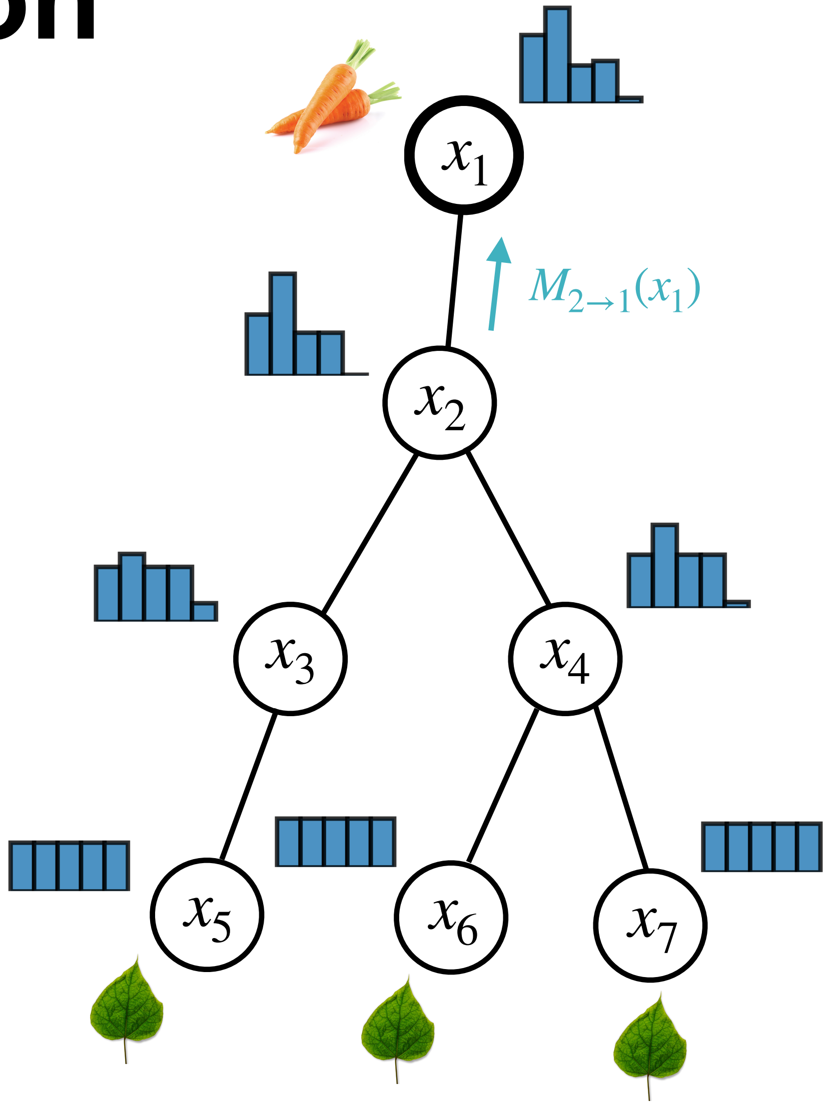
Efficient implementation

Step 3. Update the **messages** and **states** all the way up to the root



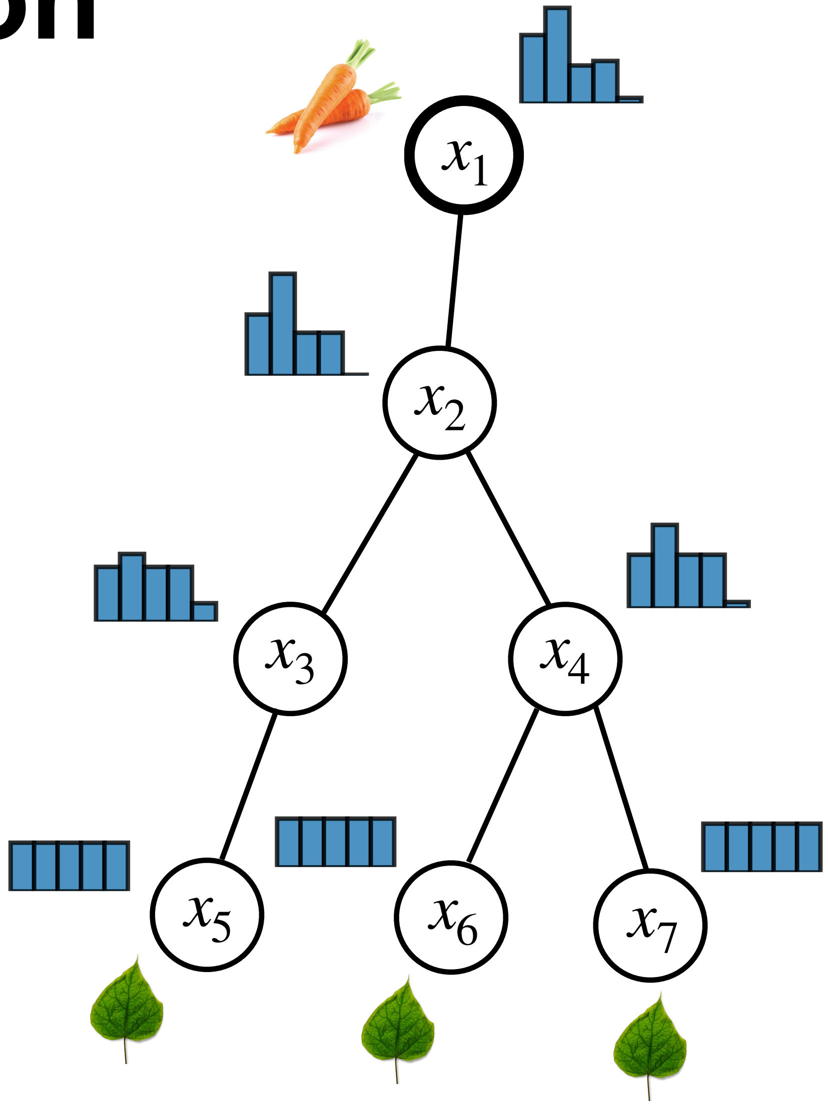
Efficient implementation

Step 3. Update the **messages** and **states** all the way up to the root



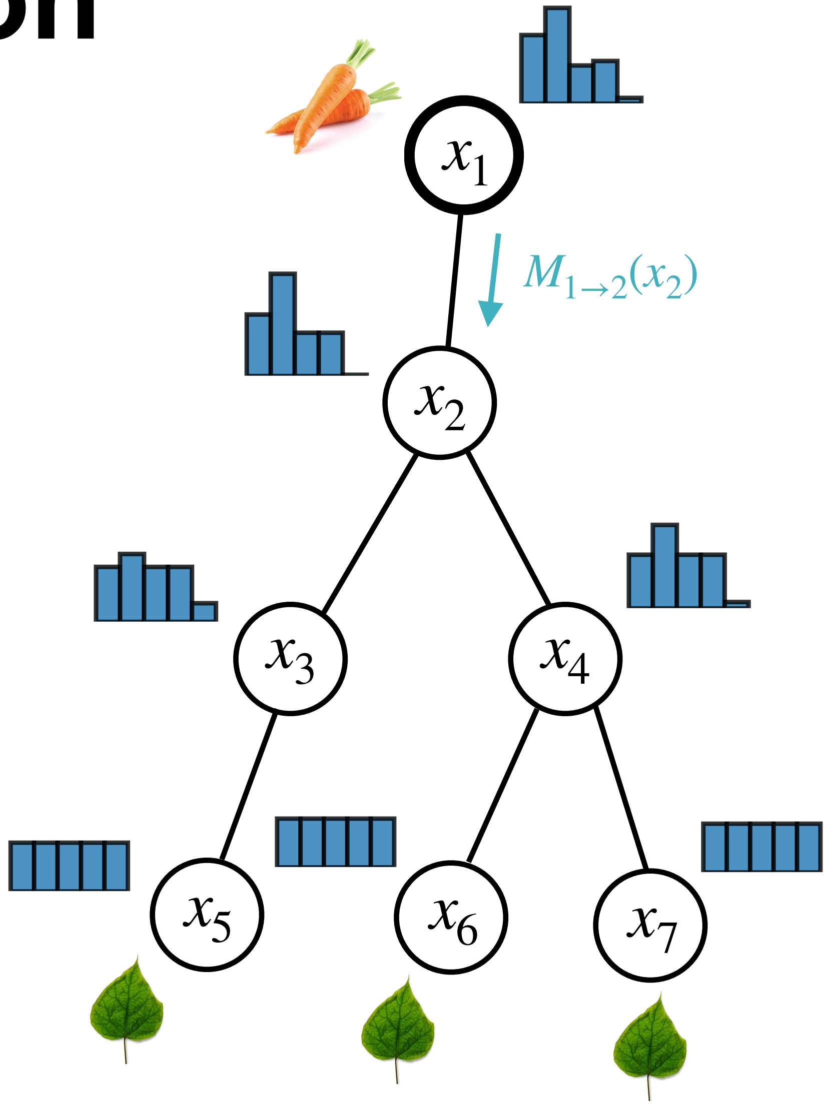
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves



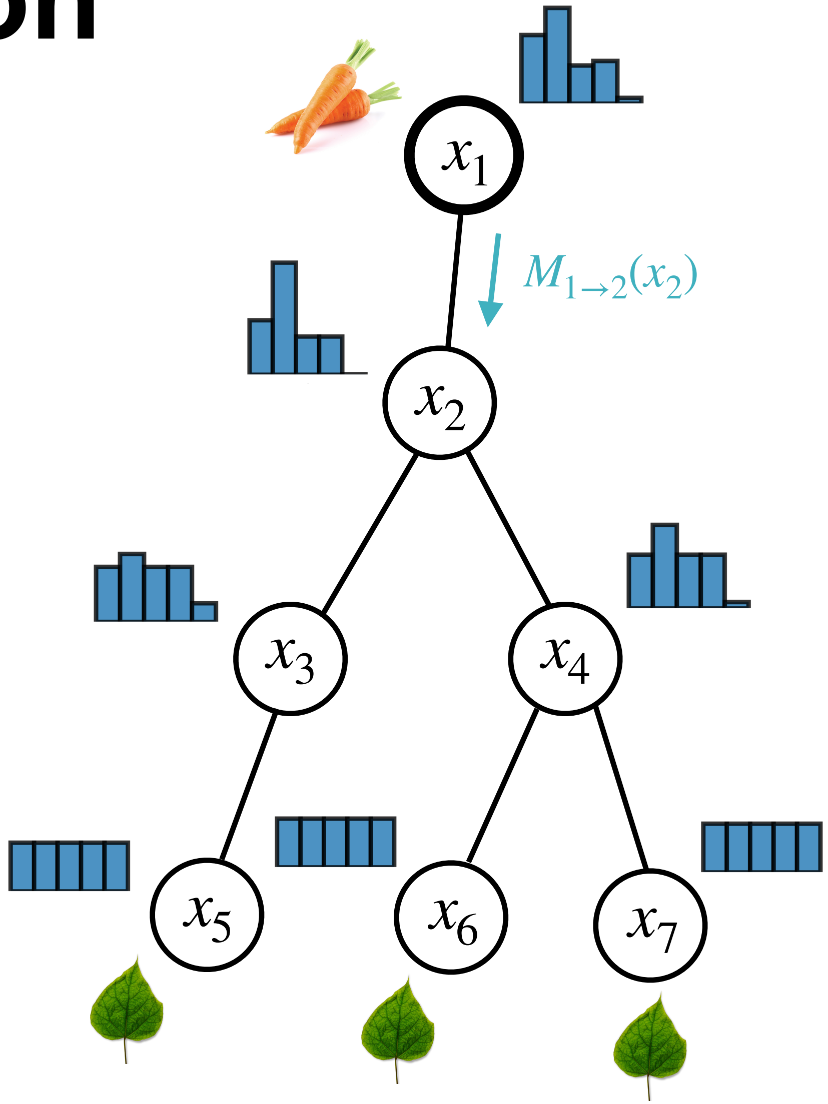
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves



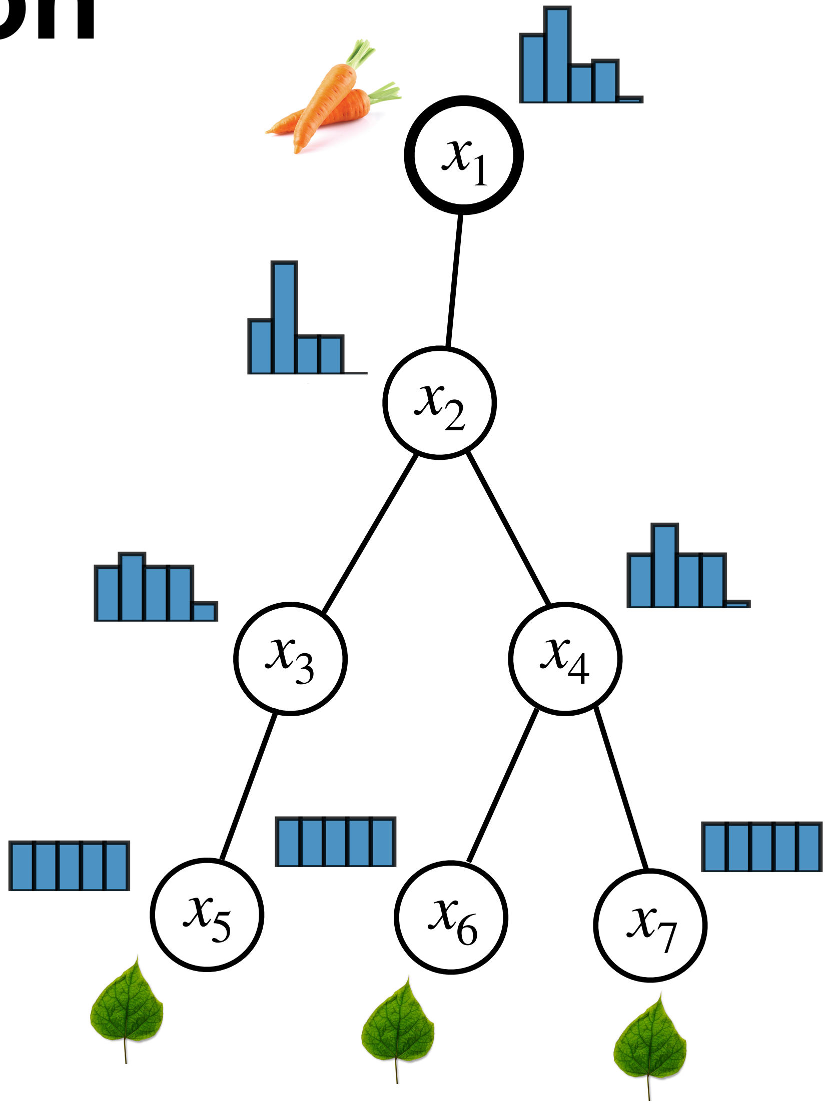
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves



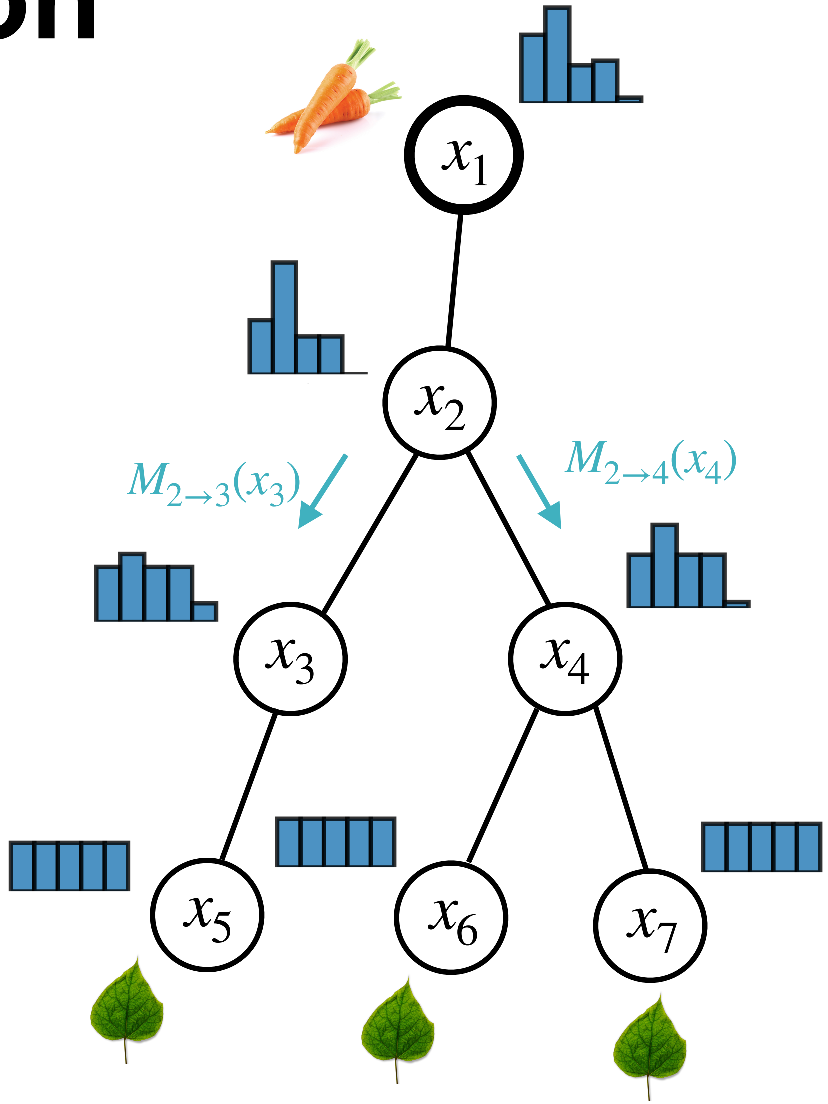
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves



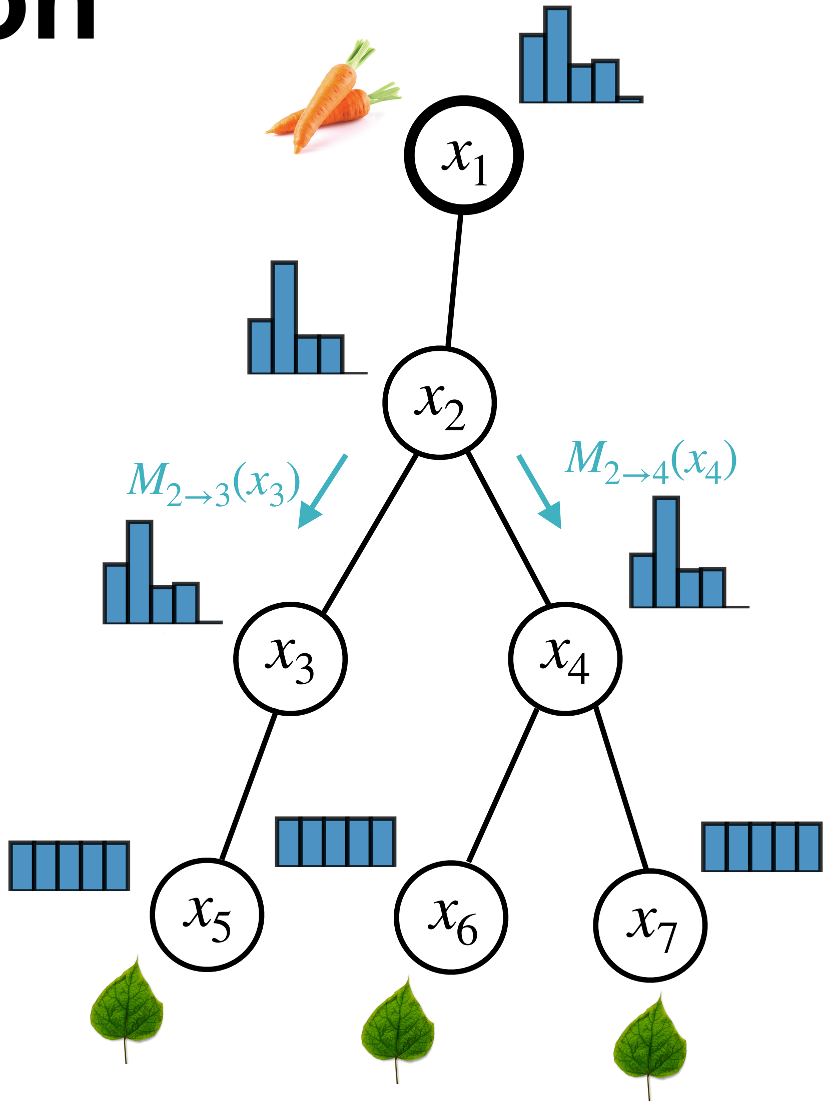
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves



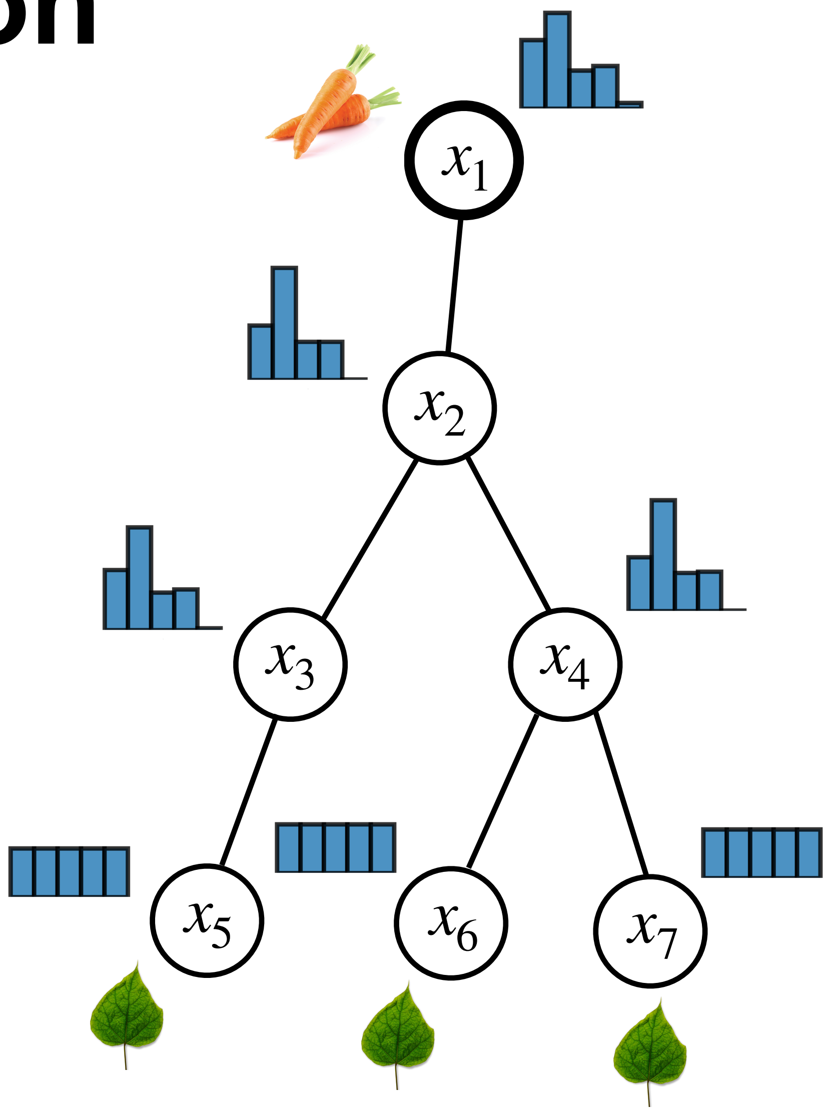
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves



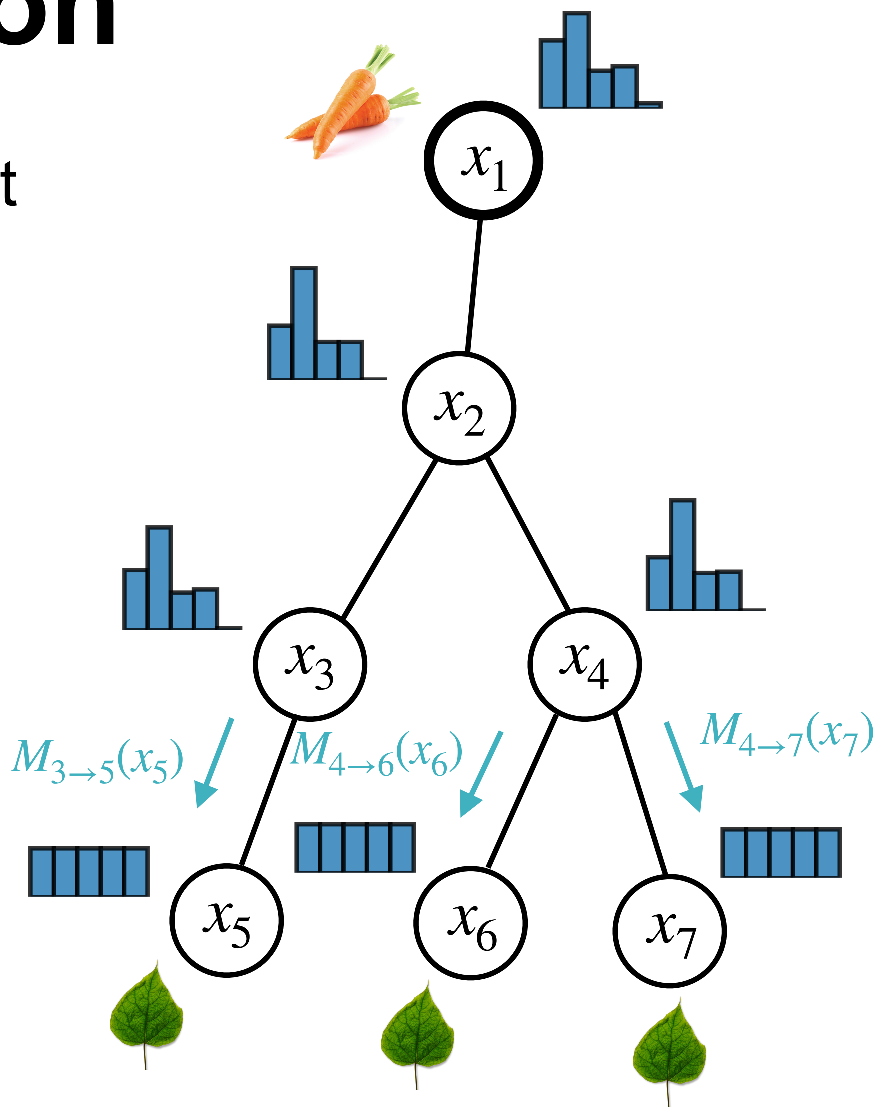
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves



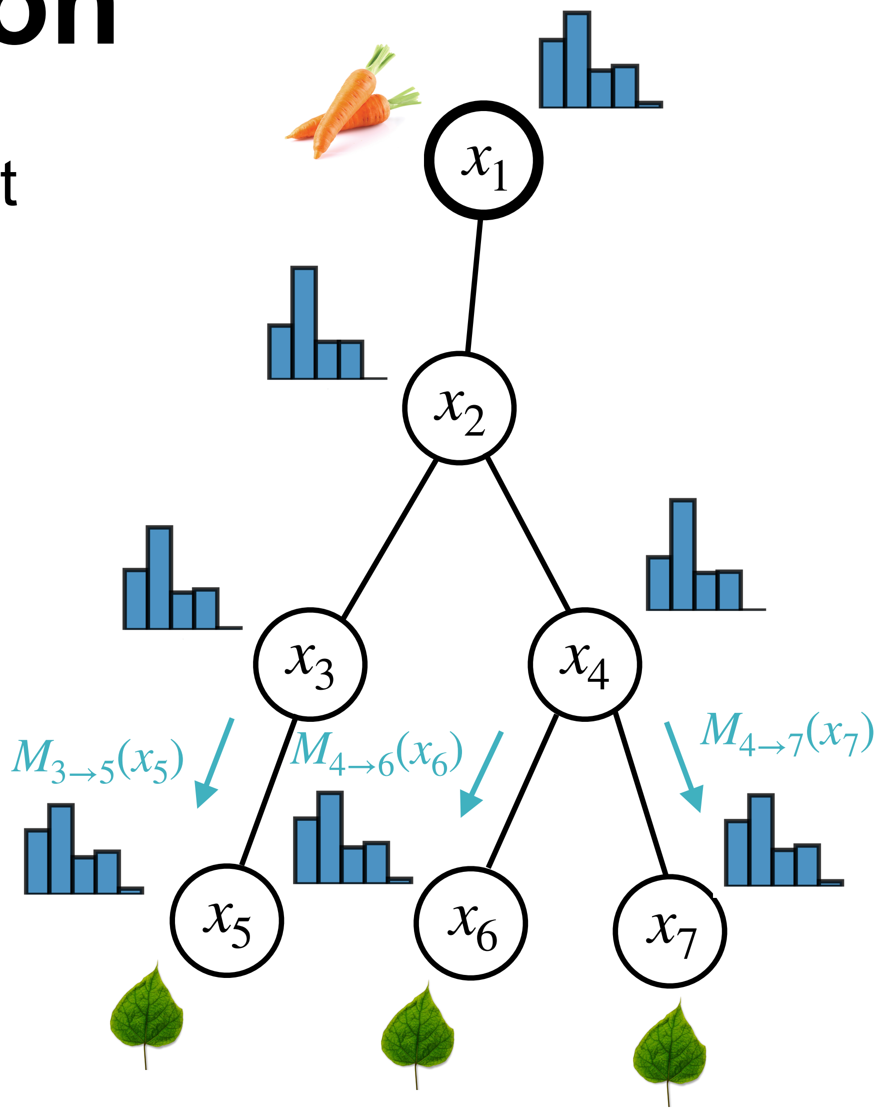
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves



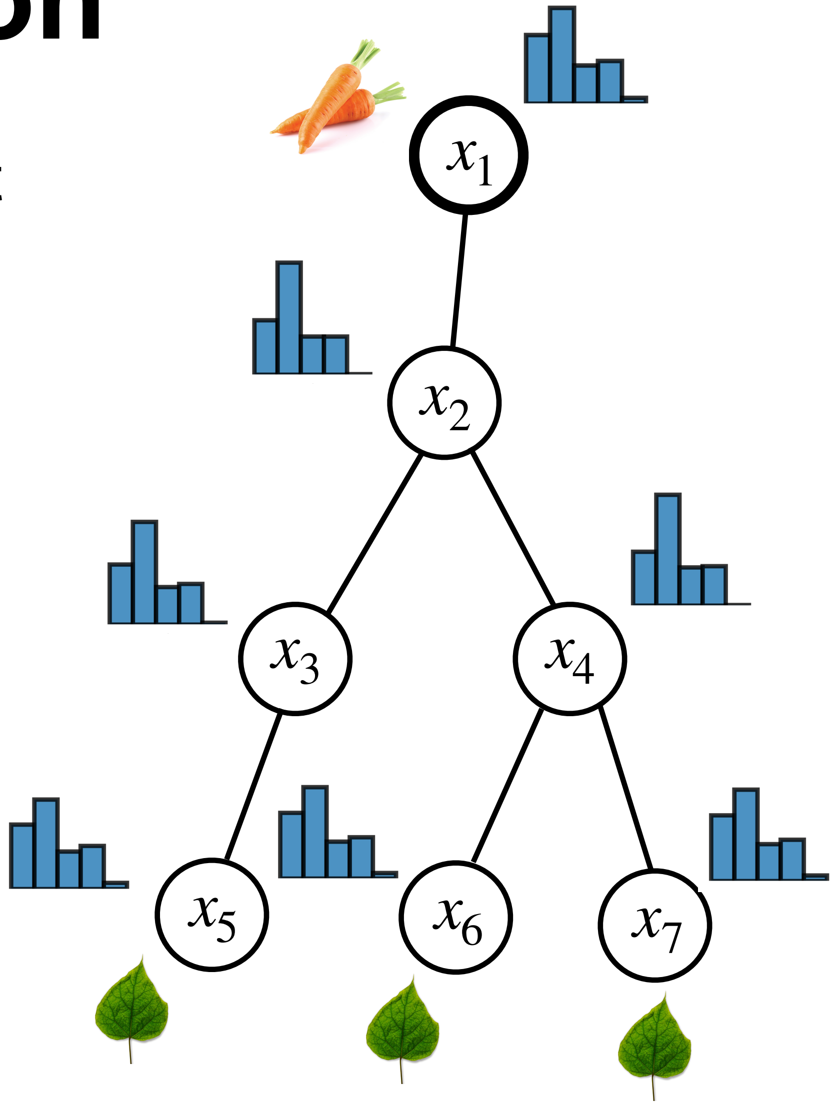
Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves

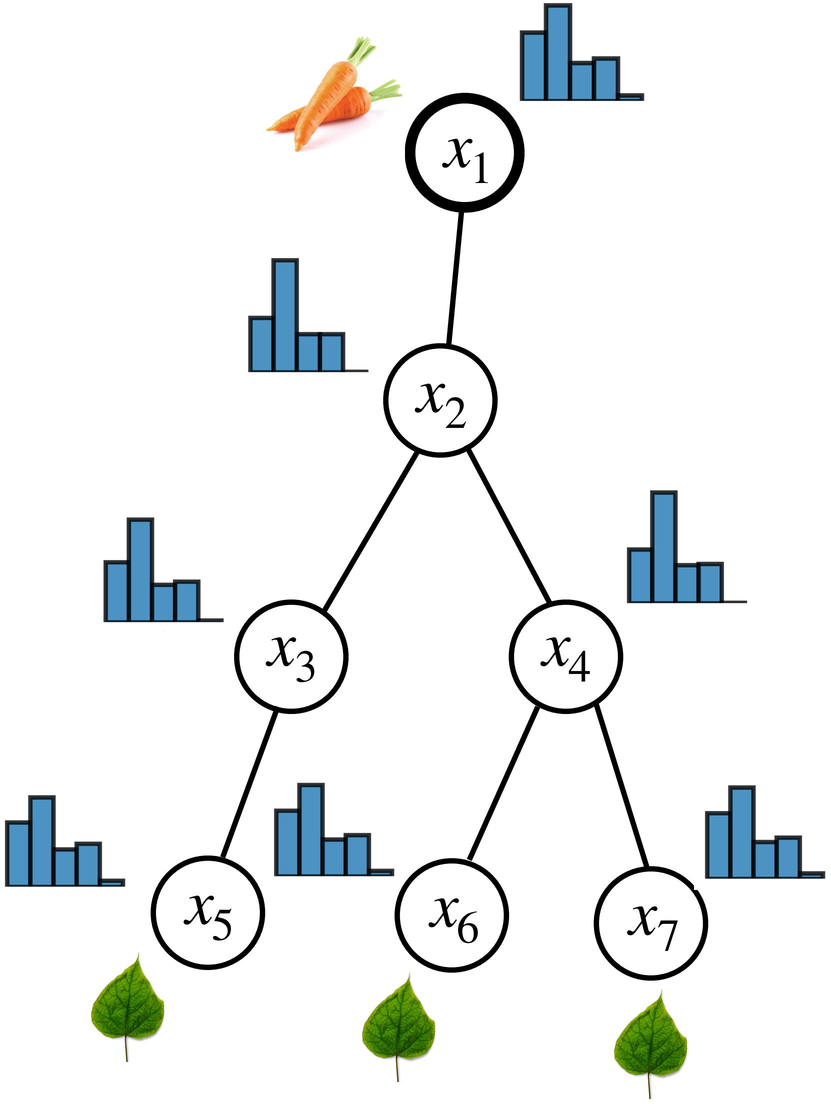


Efficient implementation

Step 4. Do the same, starting from the root and going down to the leaves

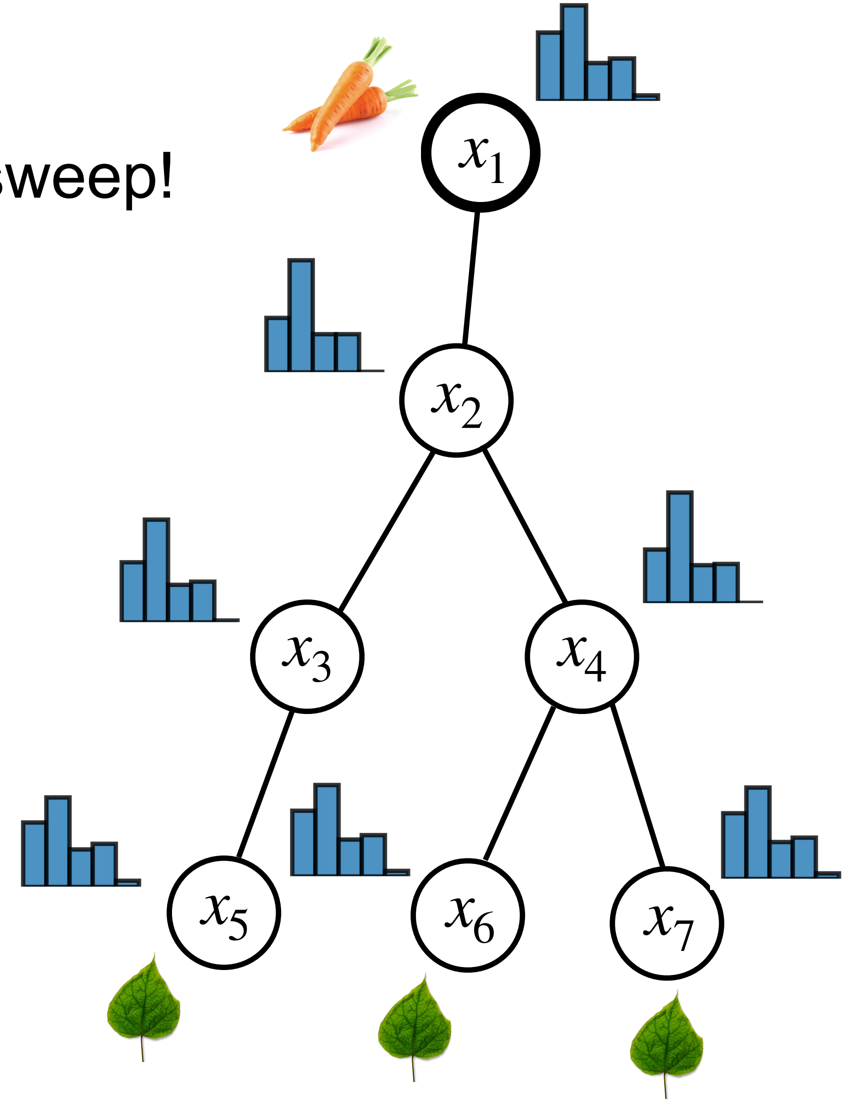


Remarks



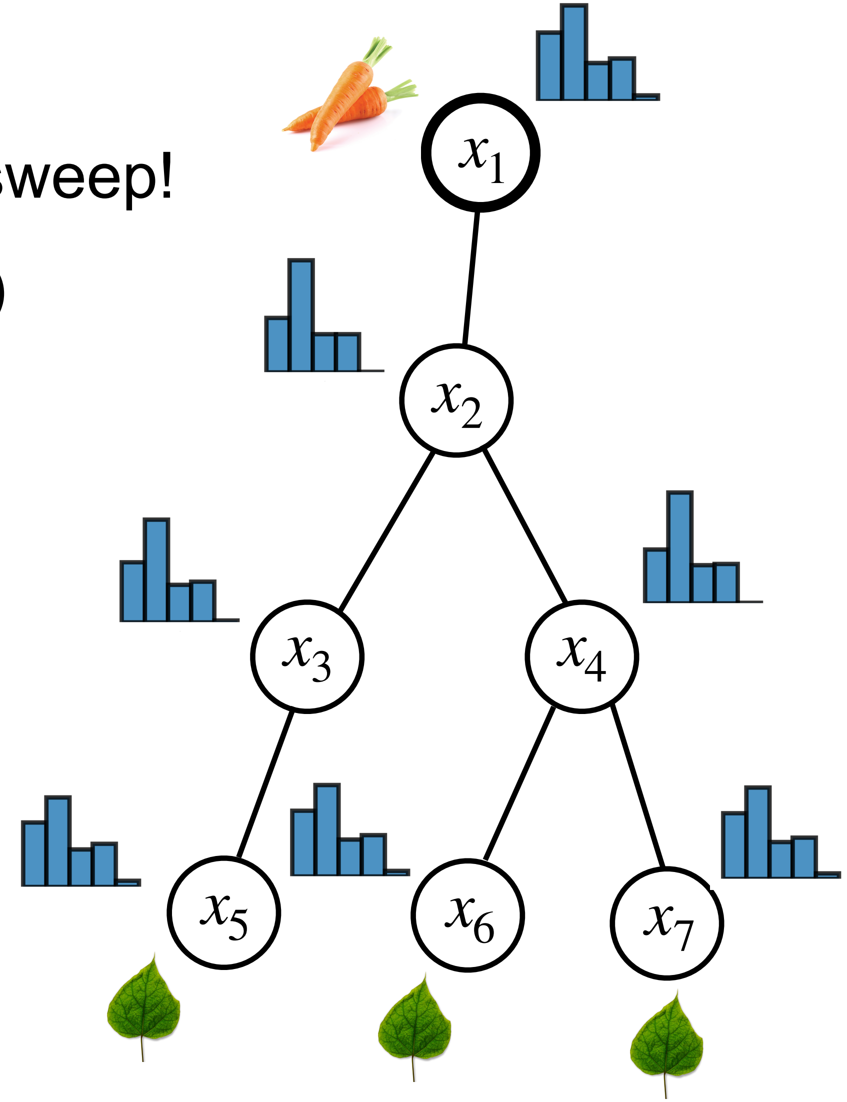
Remarks

- Guaranteed convergence after a single sweep!



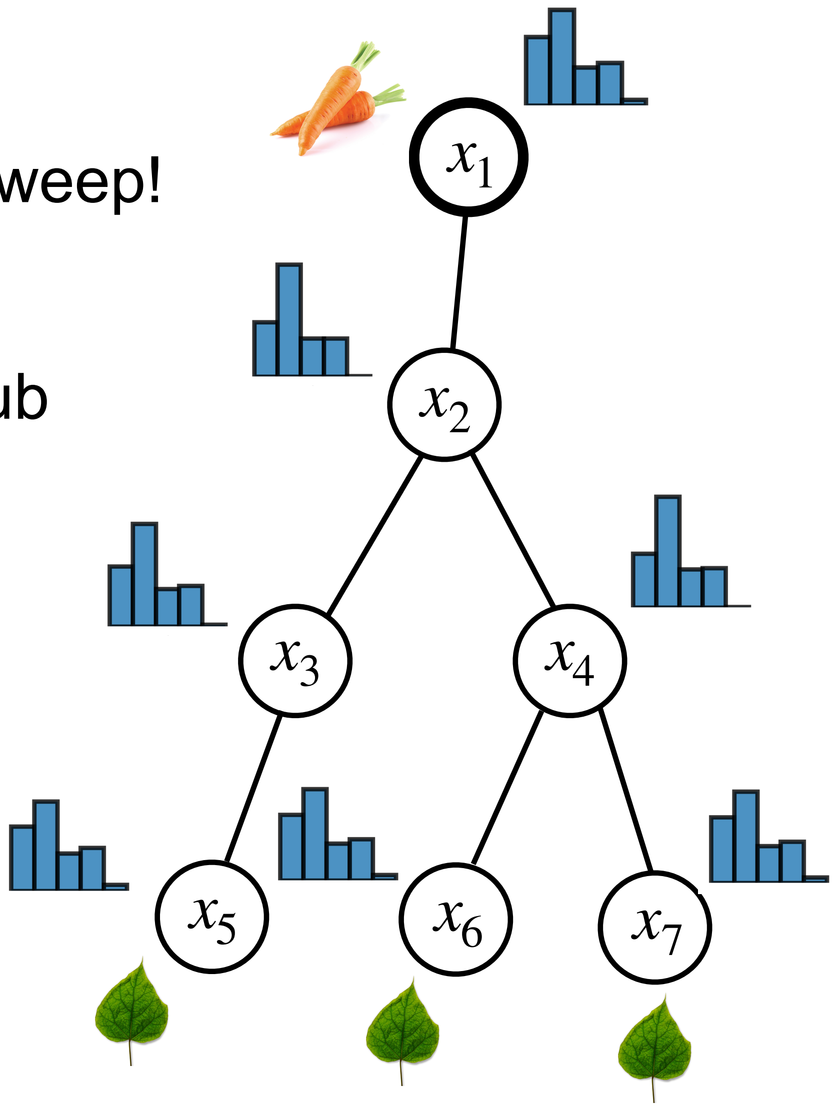
Remarks

- Guaranteed convergence after a single sweep!
- *Linear* complexity in N (**not** exponential!)



Remarks

- Guaranteed convergence after a single sweep!
- *Linear* complexity in N (**not** exponential!)
- See example implementation in my GitHub



Checklist

If $G = (V, E)$ is a tree, can we compute:

Checklist

If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data

2. The marginal distribution $p(z)$ of latent variables

Checklist

If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data ✓

2. The marginal distribution $p(z)$ of latent variables ✓

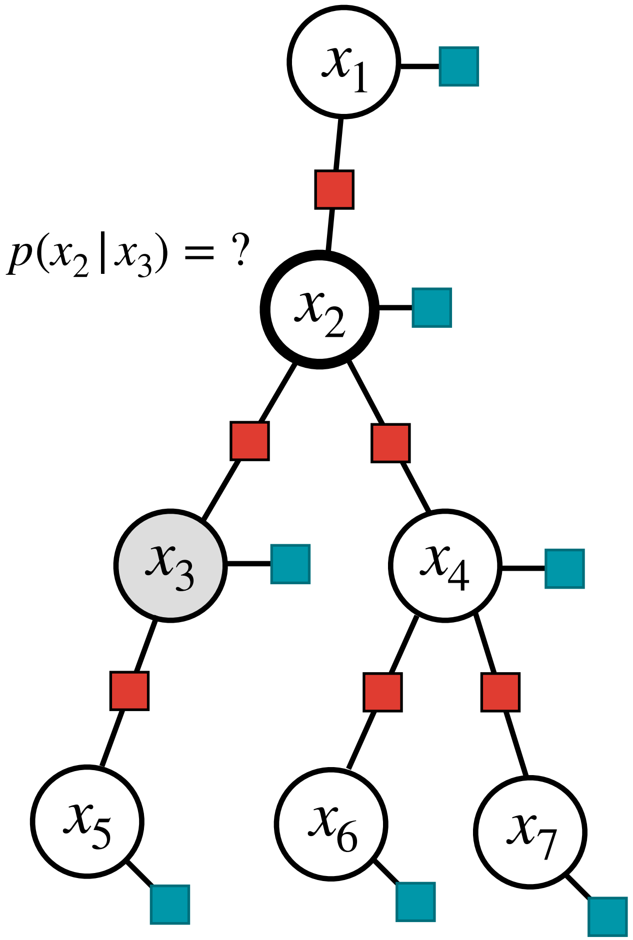
Checklist

If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data ✓

2. The marginal distribution $p(z)$ of latent variables ✓

3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$



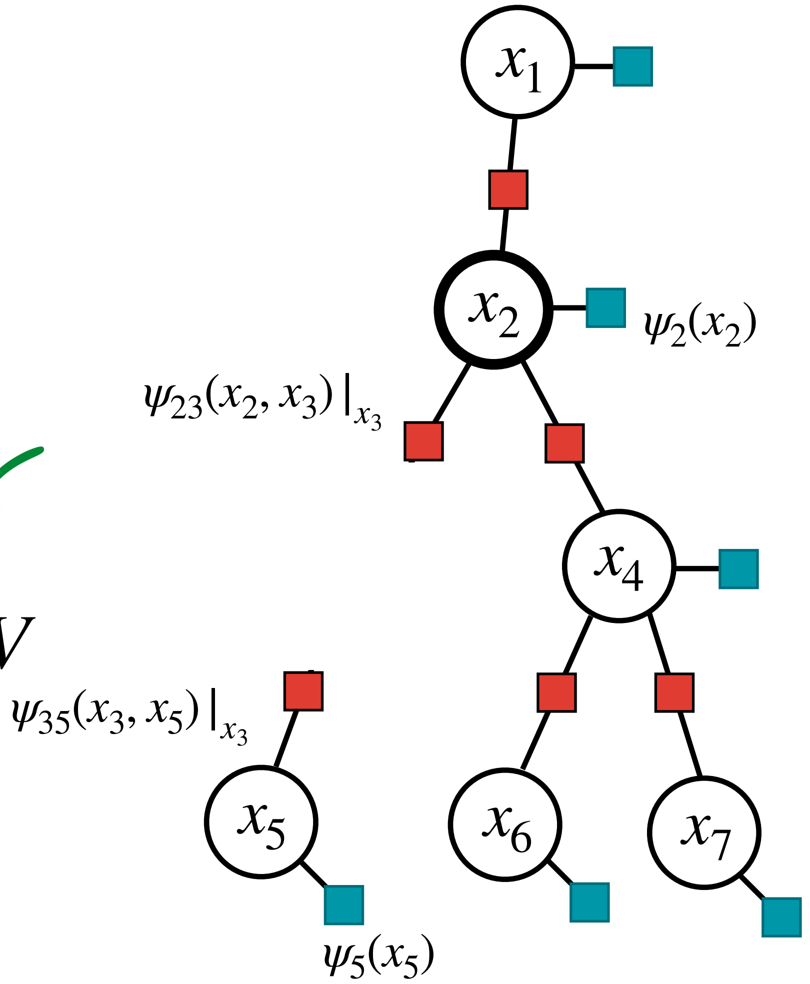
Checklist

If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data ✓

2. The marginal distribution $p(z)$ of latent variables ✓

3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$



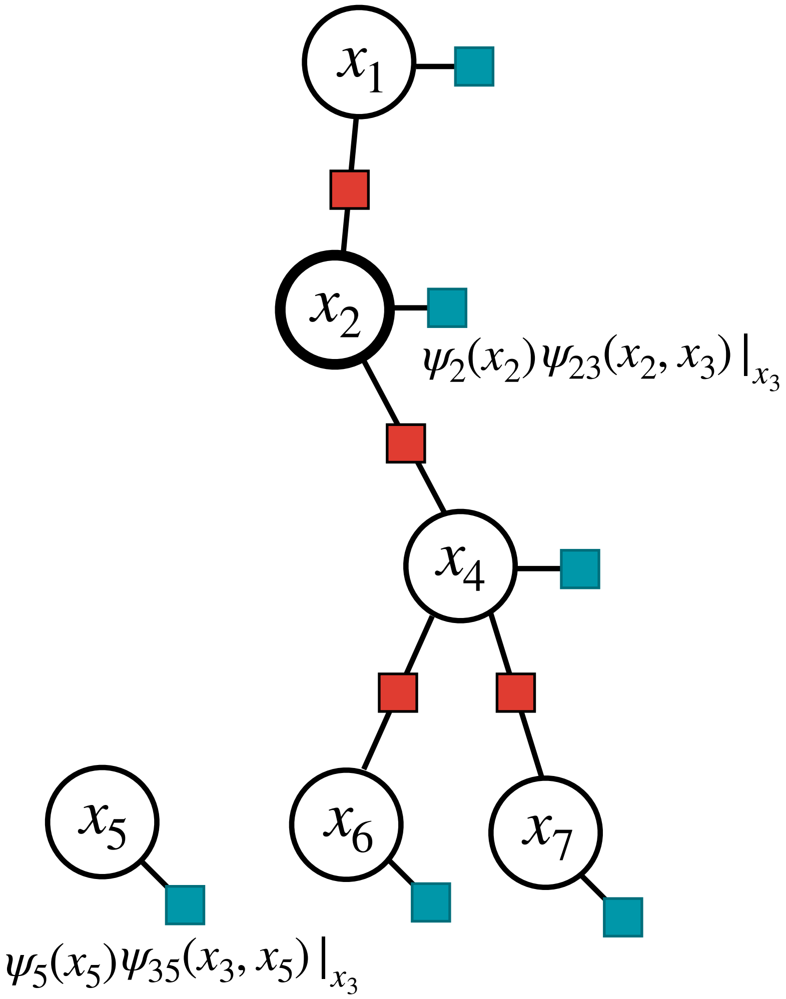
Checklist

If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data ✓

2. The marginal distribution $p(z)$ of latent variables ✓

3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$



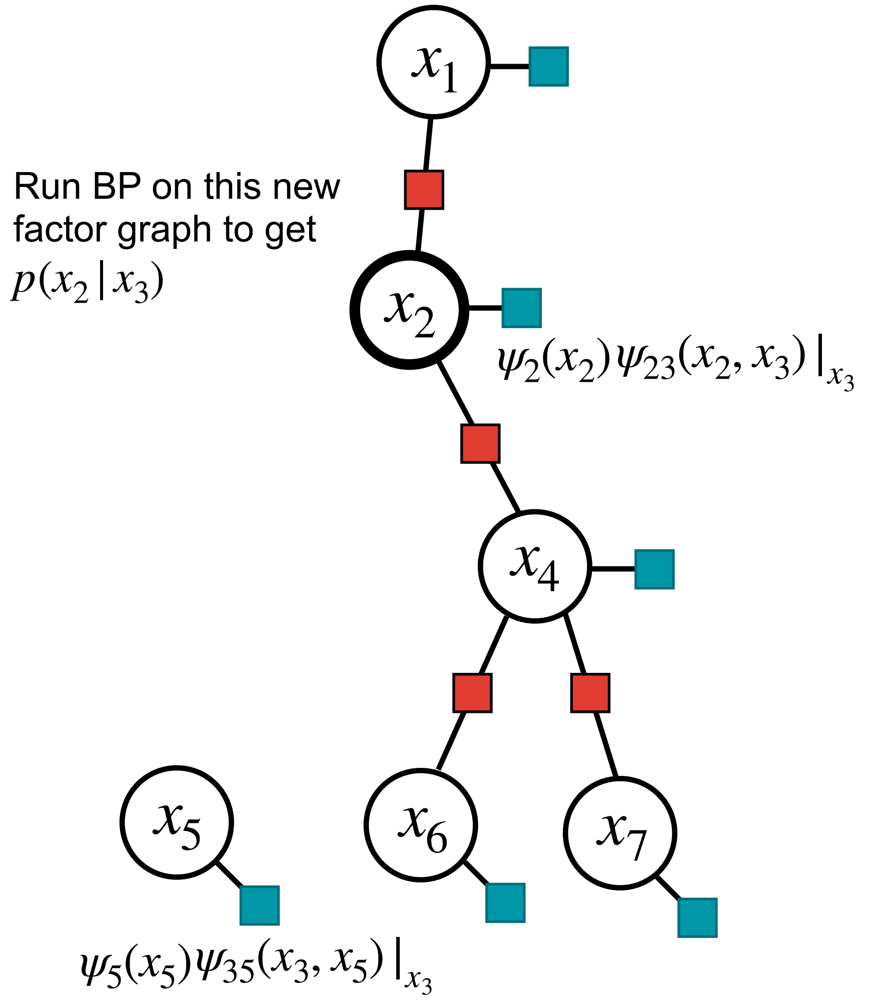
Checklist

If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data ✓

2. The marginal distribution $p(z)$ of latent variables ✓

3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$



Checklist

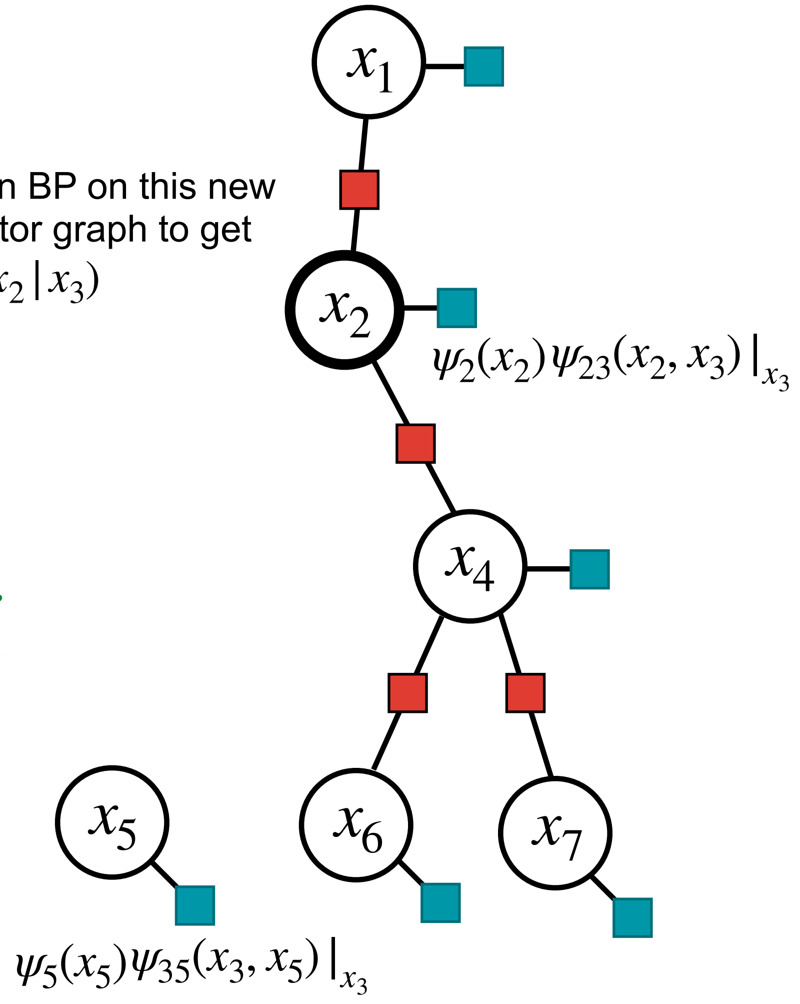
If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data ✓

2. The marginal distribution $p(z)$ of latent variables ✓

3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$ ✓

Run BP on this new factor graph to get $p(x_2 \mid x_3)$



Checklist

If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data ✓

2. The marginal distribution $p(z)$ of latent variables ✓

3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$ ✓

4. The mode $\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$

Checklist

If $G = (V, E)$ is a tree, can we compute:

1. The marginal likelihood $p(y)$ of observed data ✓

2. The marginal distribution $p(z)$ of latent variables ✓

3. The conditional distribution $p(x_i \mid x_j)$ for any $i, j \in V$ ✓

4. The mode $\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x})$ ✗

References

[1] Bishop, Christopher M. *Pattern Recognition and Machine Learning*. New York: springer, 2006.

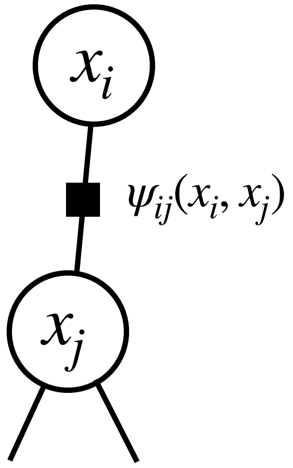
[2] Wainwright, Martin J., and Michael I. Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Foundations and Trends in Machine Learning, 2008.

3. Some Extensions of Belief Propagation

Recall the message passing protocol in BP:

Message update:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\dots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$



State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$

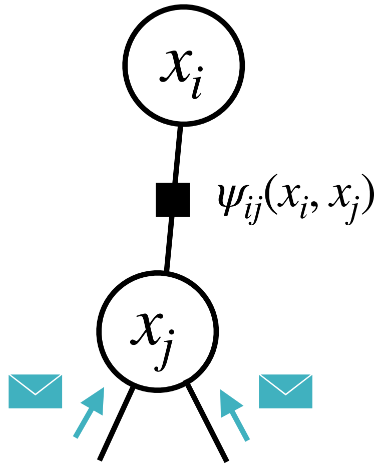
Recall the message passing protocol in BP:

Message update:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$



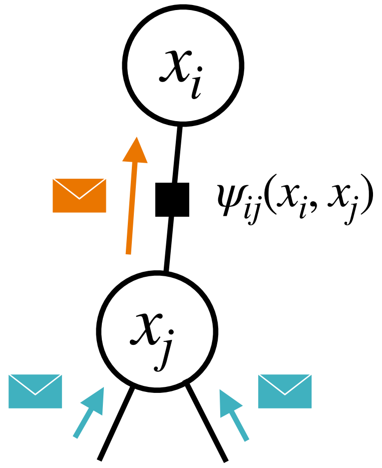
Recall the message passing protocol in BP:

Message update:

$$\boxed{M_{j\to i}(x_i)} = \sum_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\,\boxed{\psi_j(x_j)} \prod_{k\sim j, k\neq i} \boxed{M_{k\to j}(x_j)},$$

State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$



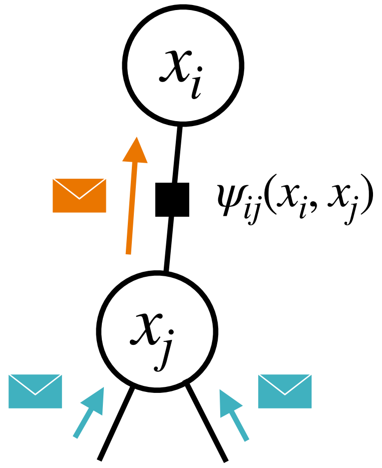
Recall the message passing protocol in BP:

Message update:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$



We assume that the graph is **tree-structured**.

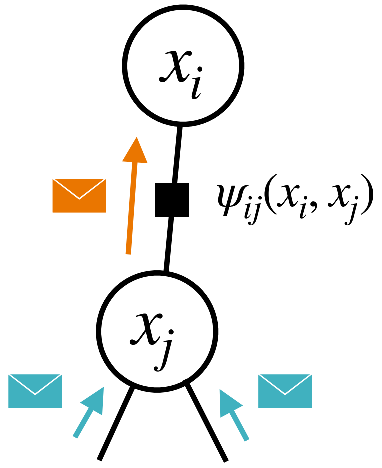
Recall the message passing protocol in BP:

Message update:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$



We assume that the graph is **tree-structured**.

What extensions can we consider?

Extension 1. Continuous states

Extension 1. Continuous states

When states are *continuous* $x_i \in \mathbb{R}^d$, we replace the sum by an integral:

Message update:

$$M_{j\to i}(x_i) = \int_{\mathbb{R}^d} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)\, \mathrm{d}x_j,$$

State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$

Extension 1. Continuous states

When states are *continuous* $x_i \in \mathbb{R}^d$, we replace the sum by an integral:

Message update:

$$M_{j\to i}(x_i) = \int_{\mathbb{R}^d} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)\, \mathrm{d}x_j,$$

State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$

Extension 1. Continuous states

When states are *continuous* $x_i \in \mathbb{R}^d$, we replace the sum by an integral:

Message update:

$$M_{j\to i}(x_i) = \int_{\mathbb{R}^d} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)\, \mathrm{d}x_j,$$

State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$

The integral is generally intractable, except in some cases.

Extension 1. Continuous states

When states are *continuous* $x_i \in \mathbb{R}^d$, we replace the sum by an integral:

Message update:

$$M_{j\to i}(x_i) = \int_{\mathbb{R}^d} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)\,\mathrm{d}x_j,$$

State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$

The integral is generally intractable, except in some cases.

For e.g. **Gaussian belief propagation**.

Gaussian belief propagation

Properties of Gaussians:

Gaussian belief propagation

Properties of Gaussians:

1. Product of two Gaussians is Gaussian:

$$\mathcal{N}(x \,|\, a, A)\, \mathcal{N}(x \,|\, b, B) = \mathcal{N}(x \,|\, c, C),$$

Gaussian belief propagation

Properties of Gaussians:

1. Product of two Gaussians is Gaussian:

$$\mathcal{N}(x \mid a, A)\, \mathcal{N}(x \mid b, B) = \mathcal{N}(x \mid c, C),$$

where $$c = C(A^{-1}a + B^{-1}b), \quad C = (A^{-1} + B^{-1})^{-1}.$$

Gaussian belief propagation

Properties of Gaussians:

1. Product of two Gaussians is Gaussian:

$$\mathcal{N}(x \mid a, A)\, \mathcal{N}(x \mid b, B) = \mathcal{N}(x \mid c, C),$$

where $\quad c = C(A^{-1}a + B^{-1}b), \quad C = (A^{-1} + B^{-1})^{-1}.$

Message update:

$$M_{j\to i}(x_i) = \int_{\mathbb{R}^d} \psi_{ij}(x_i, x_j) \boxed{\psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j)}\, \mathrm{d}x_j,$$

State update:

$$p(x_i) \propto \boxed{\psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).}$$

Gaussian belief propagation

Properties of Gaussians:

1. Product of two Gaussians is Gaussian:

$$\mathcal{N}(x \mid a, A)\, \mathcal{N}(x \mid b, B) = \mathcal{N}(x \mid c, C),$$

$$\text{where} \qquad c = C(A^{-1}a + B^{-1}b), \quad C = (A^{-1} + B^{-1})^{-1}.$$

Message update:

$$M_{j \to i}(x_i) = \int_{\mathbb{R}^d} \psi_{ij}(x_i, x_j)\, \mathcal{N}(x_j \mid a, A)\, \mathrm{d}x_j,$$

State update:

$$p(x_i) = \mathcal{N}(x_i \mid \mu_i, \Sigma_i).$$

Gaussian belief propagation

Properties of Gaussians:

2. Integral of Gaussians is Gaussian:

i.) $$\int_{\mathbb{R}^d} \mathcal{N}(x \mid Hx', R)\, \mathcal{N}(x' \mid a, A)\, \mathrm{d}x' = \mathcal{N}(x \mid Ha, HAH^T + R),$$

ii.) $$\int_{\mathbb{R}^d} \mathcal{N}(x \mid Hx', R)\, \mathcal{N}(x \mid a, A)\, \mathrm{d}x = \mathcal{N}(Hx' \mid a, A + R).$$

Gaussian belief propagation

Properties of Gaussians:

2. Integral of Gaussians is Gaussian:

i.) $$\int_{\mathbb{R}^d} \mathcal{N}(x \,|\, Hx', R)\, \mathcal{N}(x' \,|\, a, A)\, \mathrm{d}x' = \mathcal{N}(x \,|\, Ha, HAH^T + R),$$

ii.) $$\int_{\mathbb{R}^d} \mathcal{N}(x \,|\, Hx', R)\, \mathcal{N}(x \,|\, a, A)\, \mathrm{d}x = \mathcal{N}(Hx' \,|\, a, A + R).$$

Message update:

$$M_{j \to i}(x_i) = \boxed{\int_{\mathbb{R}^d} \psi_{ij}(x_i, x_j)\, \mathcal{N}(x_j \,|\, a, A)\, \mathrm{d}x_j,}$$

State update:

$$p(x_i) = \mathcal{N}(x_i \,|\, \mu_i, \Sigma_i).$$

Gaussian belief propagation

Properties of Gaussians:

2. Integral of Gaussians is Gaussian:

i.) $$\int_{\mathbb{R}^d} \mathcal{N}(x \,|\, Hx', R)\, \mathcal{N}(x' \,|\, a, A)\, \mathrm{d}x' = \mathcal{N}(x \,|\, Ha, HAH^T + R),$$

ii.) $$\int_{\mathbb{R}^d} \mathcal{N}(x \,|\, Hx', R)\, \mathcal{N}(x \,|\, a, A)\, \mathrm{d}x = \mathcal{N}(Hx' \,|\, a, A + R).$$

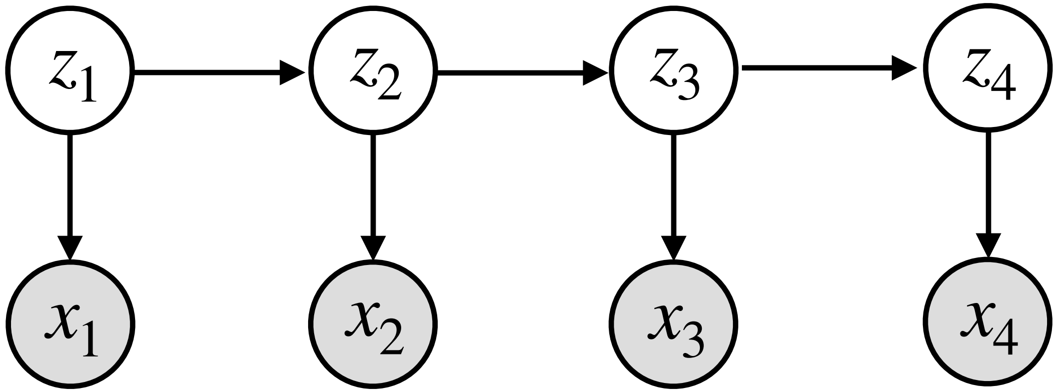
Message update:

$$M_{j\to i}(x_i) = \mathcal{N}(x_i \,|\, \mu_{j\to i}, \Sigma_{j\to i}),$$

State update:

$$p(x_i) = \mathcal{N}(x_i \,|\, \mu_i, \Sigma_i).$$

Example: Timeseries modelling

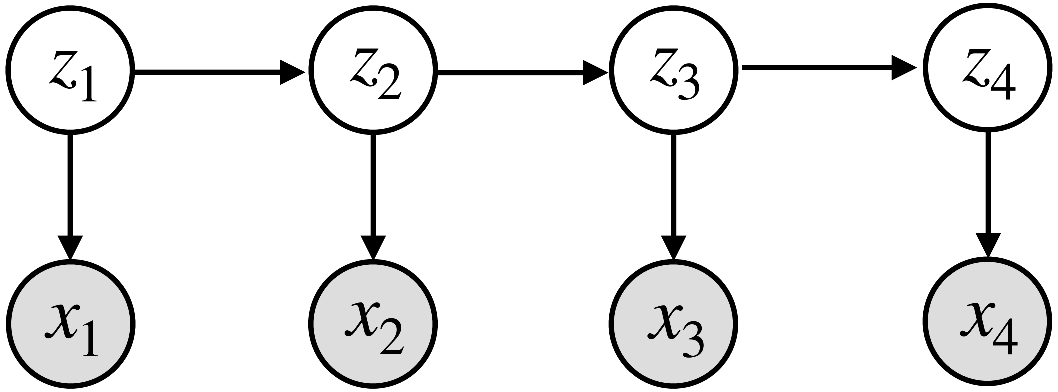


Bayesian network representation of a state-space model

Example: Timeseries modelling

Consider a linear state-space model:

$$z_{n+1} = Mz_n + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, Q),$$
$$x_n = Hz_n + \eta_n, \quad \eta_n \sim \mathcal{N}(0, R)\,.$$



Bayesian network representation of a state-space model

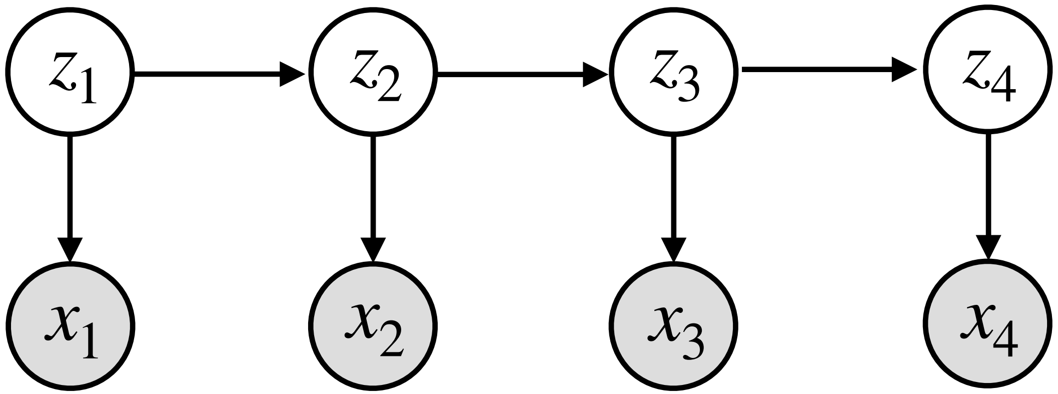
Example: Timeseries modelling

Consider a linear state-space model:

$$z_{n+1} = Mz_n + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, Q),$$
$$x_n = Hz_n + \eta_n, \quad \eta_n \sim \mathcal{N}(0, R).$$

Equivalently,

$$p(z_{n+1} \mid z_n) = \mathcal{N}(z_{n+1} \mid Mz_n, Q),$$
$$p(x_n \mid z_n) = \mathcal{N}(x_n \mid Hz_n, R).$$



Bayesian network representation of a state-space model

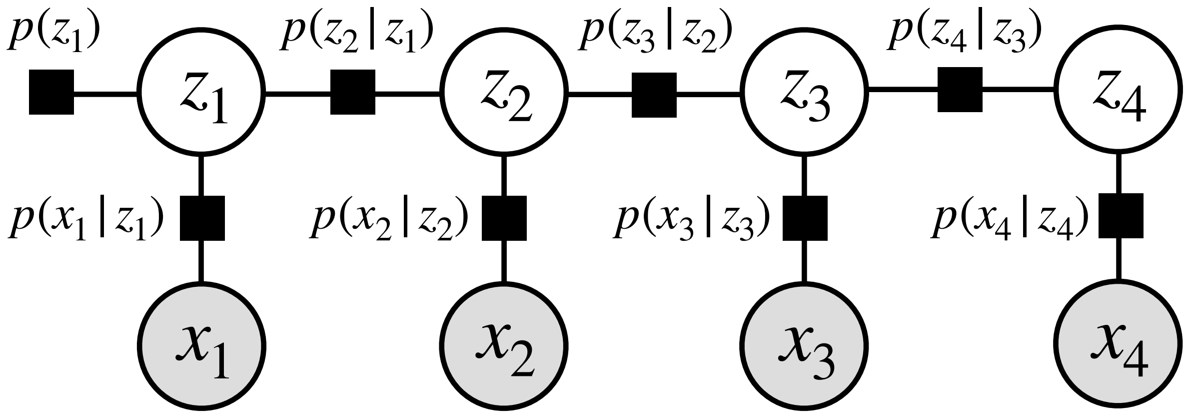
Example: Timeseries modelling

Consider a linear state-space model:

$$z_{n+1} = Mz_n + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, Q),$$
$$x_n = Hz_n + \eta_n, \quad \eta_n \sim \mathcal{N}(0, R)\,.$$

Equivalently,

$$p(z_{n+1} \,|\, z_n) = \mathcal{N}(z_{n+1} \,|\, Mz_n, Q),$$
$$p(x_n \,|\, z_n) = \mathcal{N}(x_n \,|\, Hz_n, R)\,.$$



Factor graph representation of a state-space model

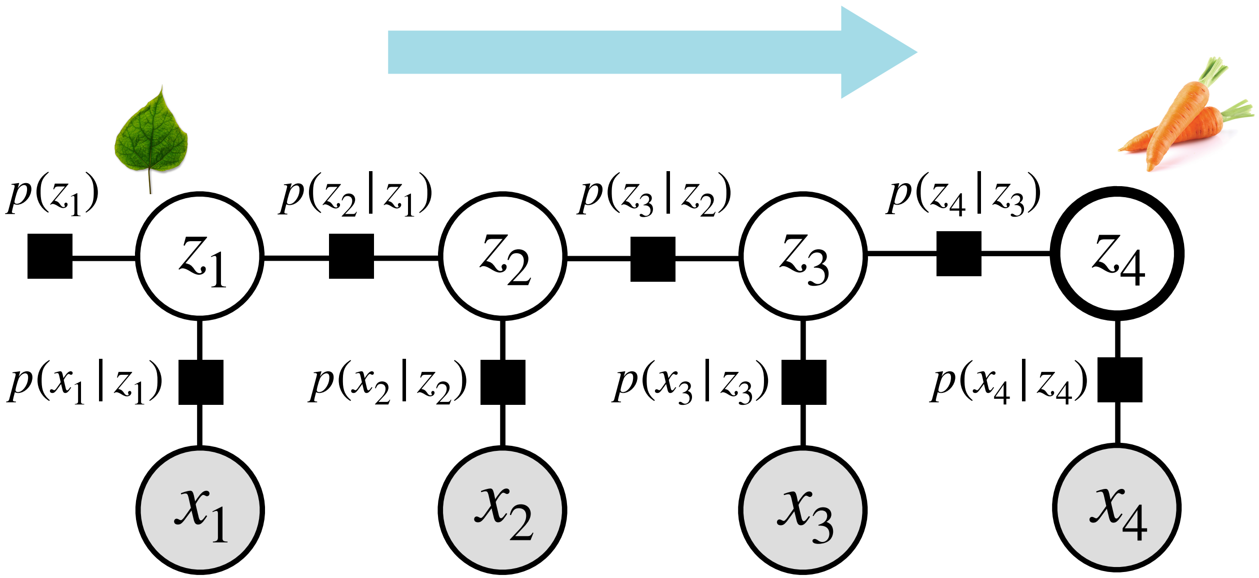
Example: Timeseries modelling

Consider a linear state-space model:

$$
\begin{aligned}
z_{n+1} &= M z_n + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, Q), \\
x_n &= H z_n + \eta_n, \quad \eta_n \sim \mathcal{N}(0, R)\,.
\end{aligned}
$$

Equivalently,

$$
\begin{aligned}
p(z_{n+1} \,|\, z_n) &= \mathcal{N}(z_{n+1} \,|\, M z_n, Q), \\
p(x_n \,|\, z_n) &= \mathcal{N}(x_n \,|\, H z_n, R)\,.
\end{aligned}
$$



Forward sweep ≡ Kalman filter

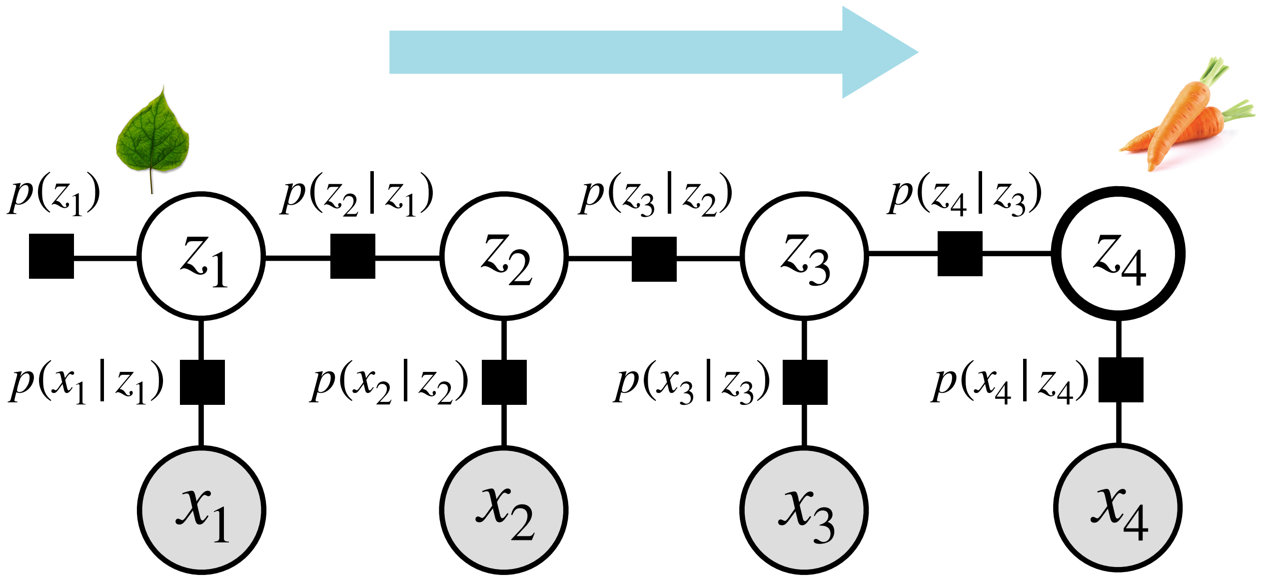
Example: Timeseries modelling

Consider a linear state-space model:

$$z_{n+1} = Mz_n + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, Q),$$
$$x_n = Hz_n + \eta_n, \quad \eta_n \sim \mathcal{N}(0, R)\,.$$

Equivalently,

$$p(z_{n+1} \,|\, z_n) = \mathcal{N}(z_{n+1} \,|\, Mz_n, Q),$$
$$p(x_n \,|\, z_n) = \mathcal{N}(x_n \,|\, Hz_n, R)\,.$$



Forward sweep ≡ Kalman filter

- Running only the forward sweep of BP is equivalent to the *Kalman filter*

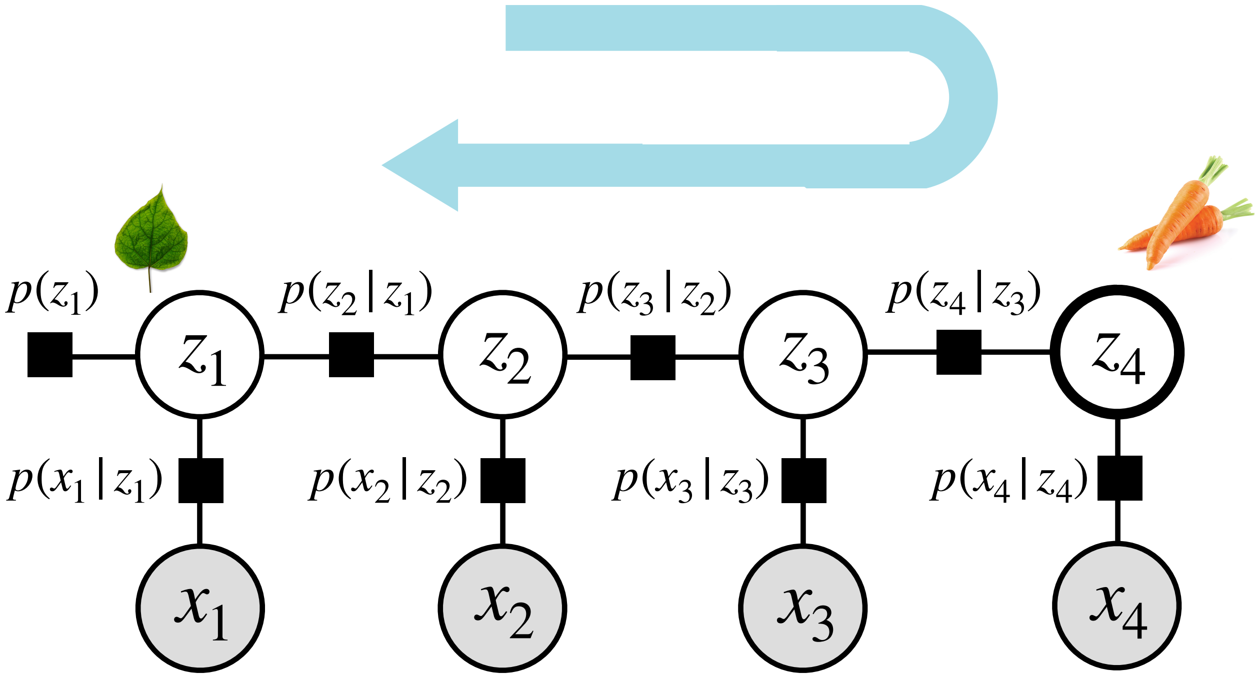
Example: Timeseries modelling

Consider a linear state-space model:

$$z_{n+1} = Mz_n + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, Q),$$
$$x_n = Hz_n + \eta_n, \quad \eta_n \sim \mathcal{N}(0, R)\,.$$

Equivalently,

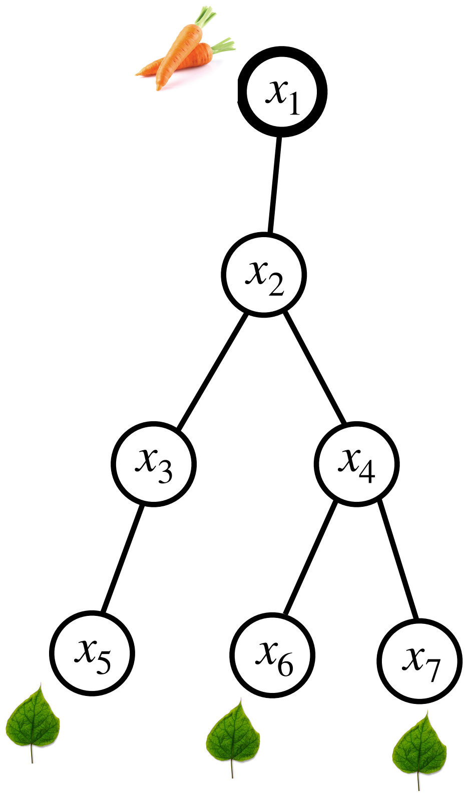
$$p(z_{n+1} \,|\, z_n) = \mathcal{N}(z_{n+1} \,|\, Mz_n, Q),$$
$$p(x_n \,|\, z_n) = \mathcal{N}(x_n \,|\, Hz_n, R)\,.$$



Forward-backward sweep ≡ RTS smoother

- Running only the forward sweep of BP is equivalent to the *Kalman filter*
- Running a full BP is equivalent to the *Rauch-Tung Striebel smoother*

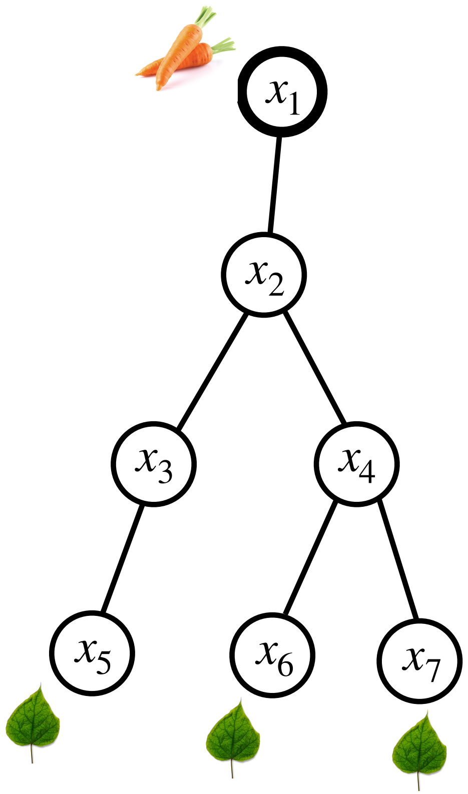
Extension 2. Max-product algorithm



Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

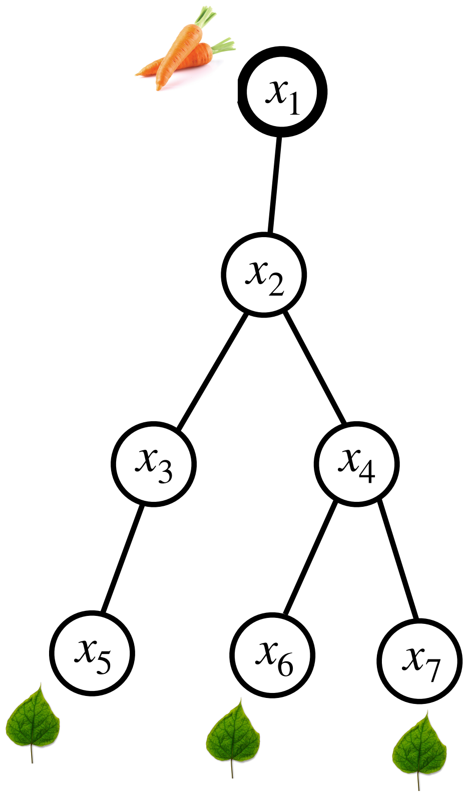
$$M_{j\to i}(x_i) = \max_{x_j\in\{1,\dots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$



Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \boxed{\max_{x_j\in\{1,\ldots,K\}}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

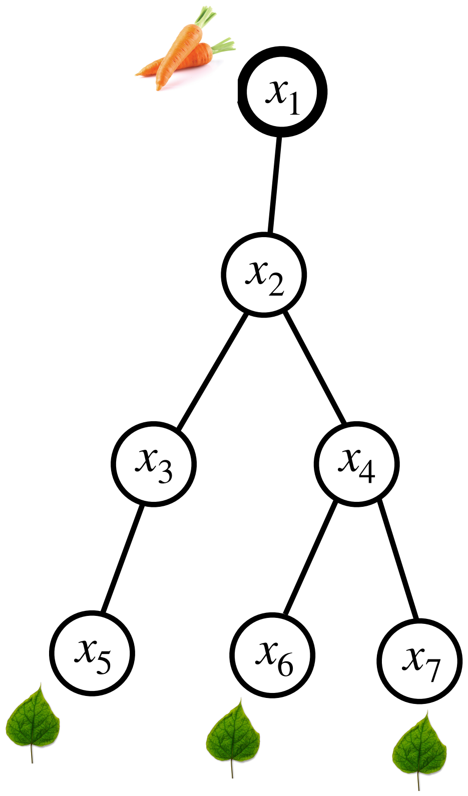


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.

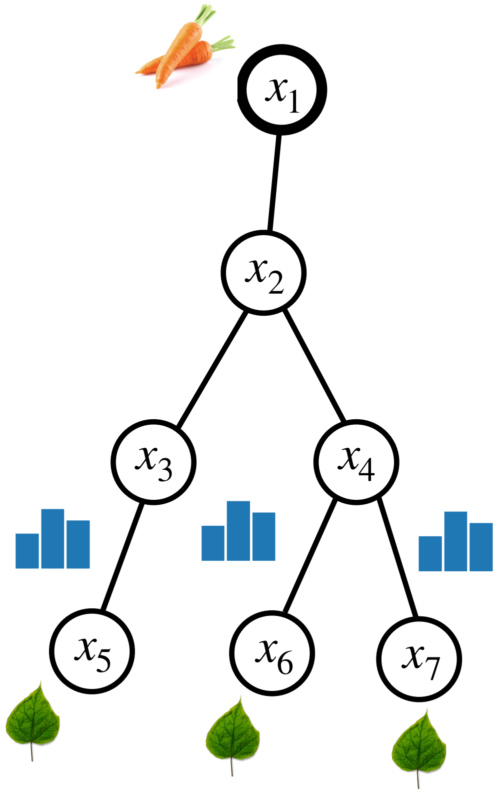


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j \to i}(x_i) = \max_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k \sim j, k \neq i} M_{k \to j}(x_j),$$

Iterate from leaf nodes up to the root node.

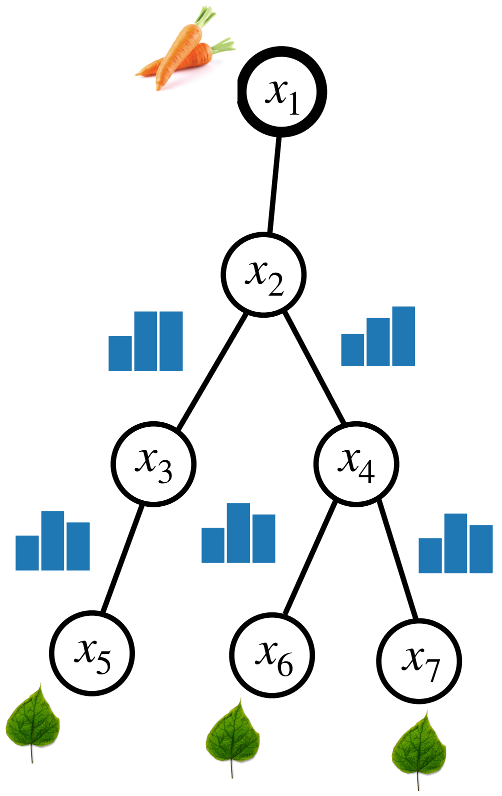


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\,\psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.

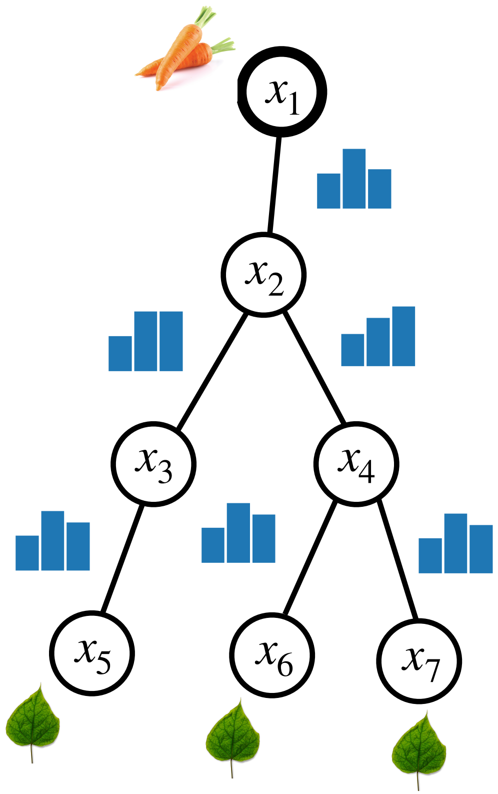


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\,\psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.

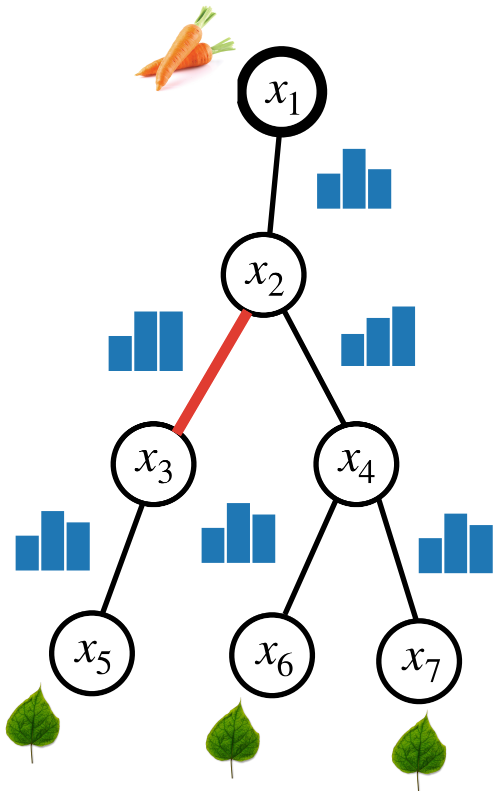


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.

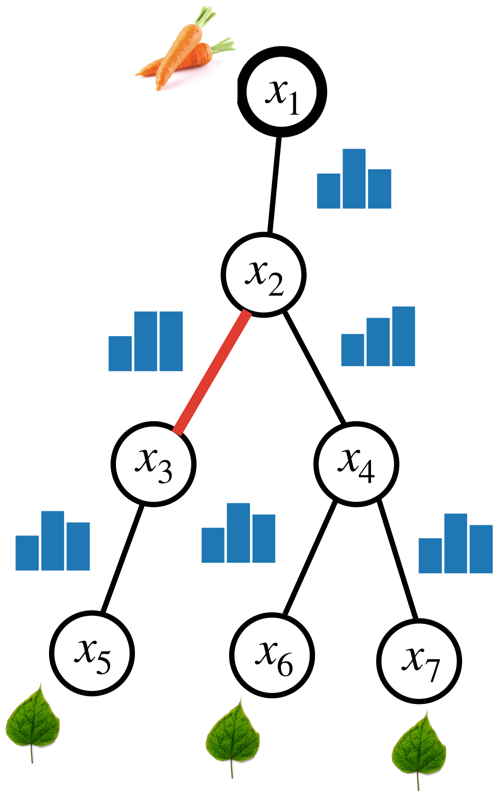


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.



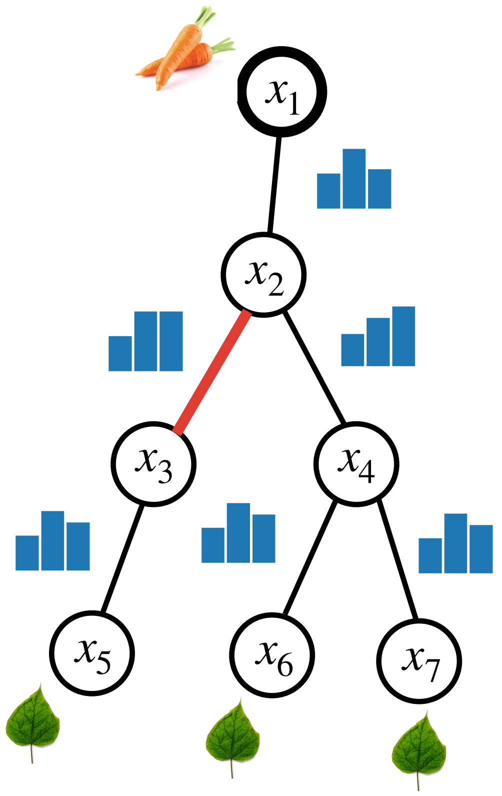
Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j \in \{1,\dots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

$=$ 

Iterate from leaf nodes up to the root node.

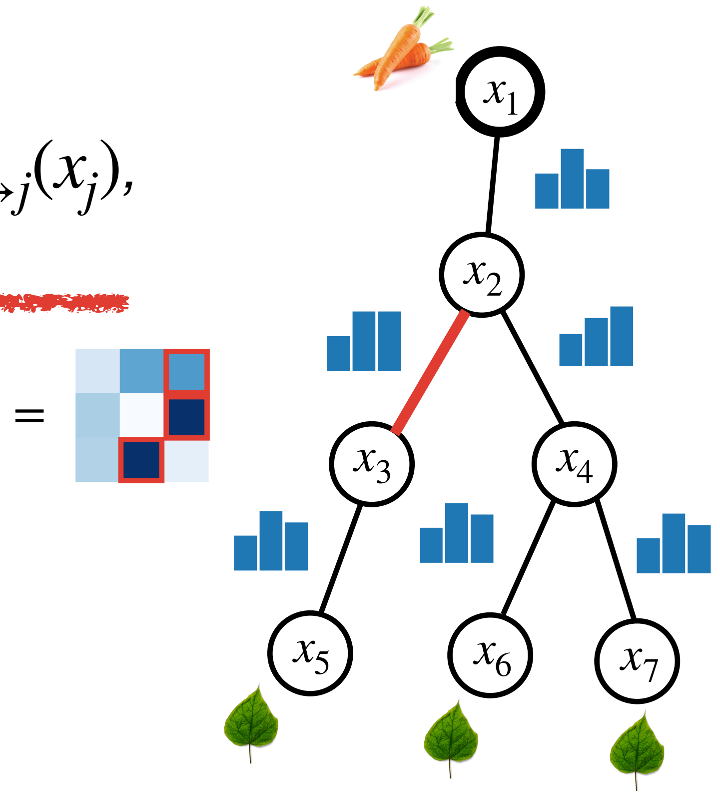


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\,\psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.

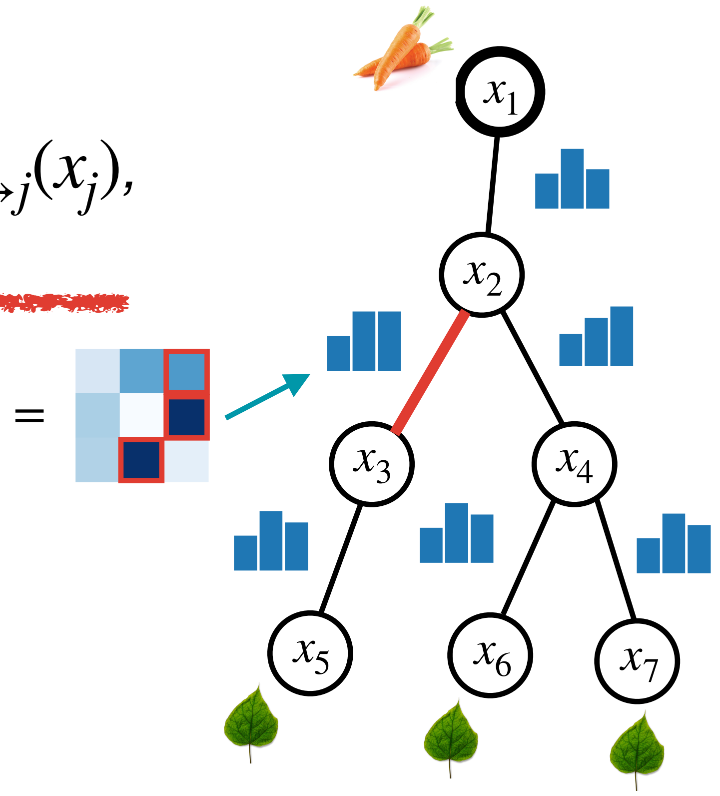


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.

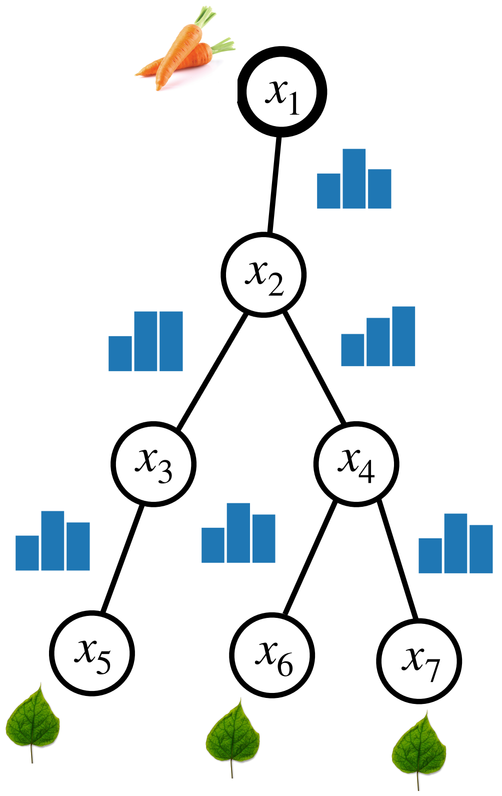


Extension 2. Max-product algorithm

Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.



Extension 2. Max-product algorithm

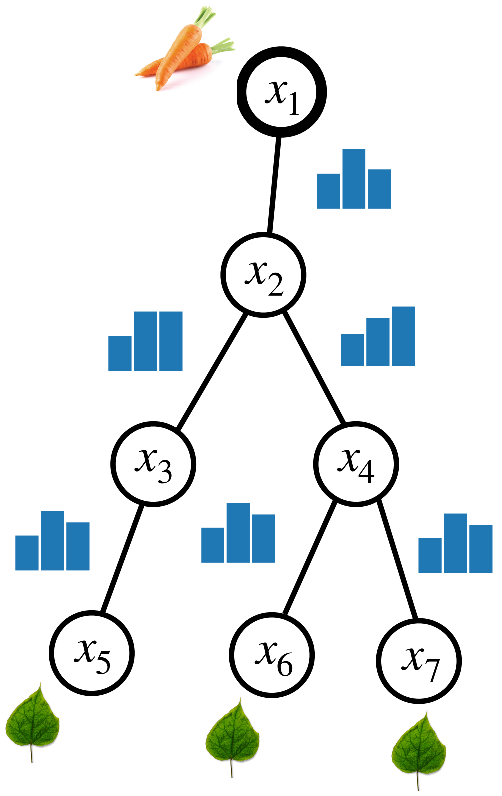
Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.

Then, we get

$$\max_{\boldsymbol{x}} p(\boldsymbol{x}) = \max_{x_{\text{root}}\in\{1,\ldots,K\}} \frac{1}{Z} \prod_{j\sim\text{root}} M_{j\to\text{root}}(x_{\text{root}}).$$



Extension 2. Max-product algorithm

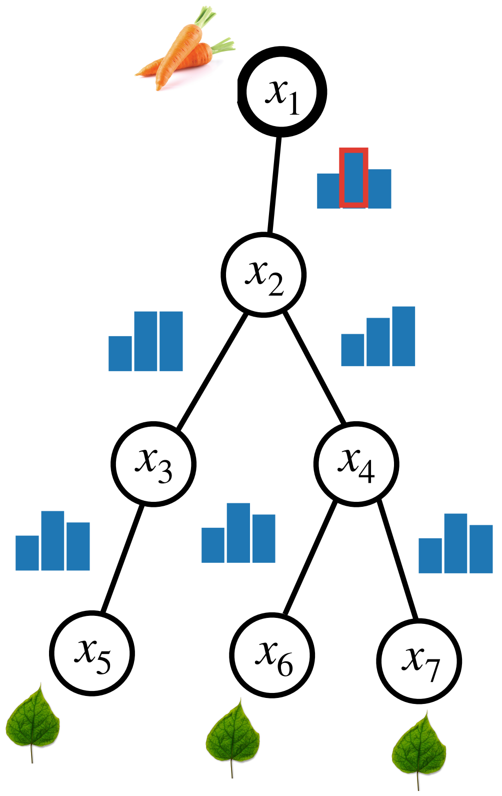
Replacing the sum in the message update by a $\max$ operator, we obtain the **max-product algorithm**:

$$M_{j\to i}(x_i) = \max_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

Iterate from leaf nodes up to the root node.

Then, we get

$$\max_{\boldsymbol{x}} p(\boldsymbol{x}) = \max_{x_{\text{root}} \in \{1,\ldots,K\}} \frac{1}{Z} \prod_{j\sim \text{root}} M_{j\to \text{root}}(x_{\text{root}}).$$

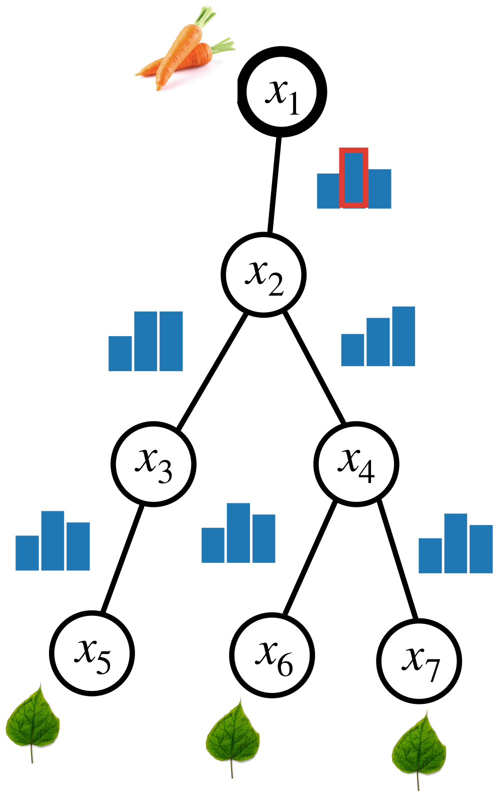


Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

using a procedure called *back-tracking:*



Extension 2. Max-product algorithm

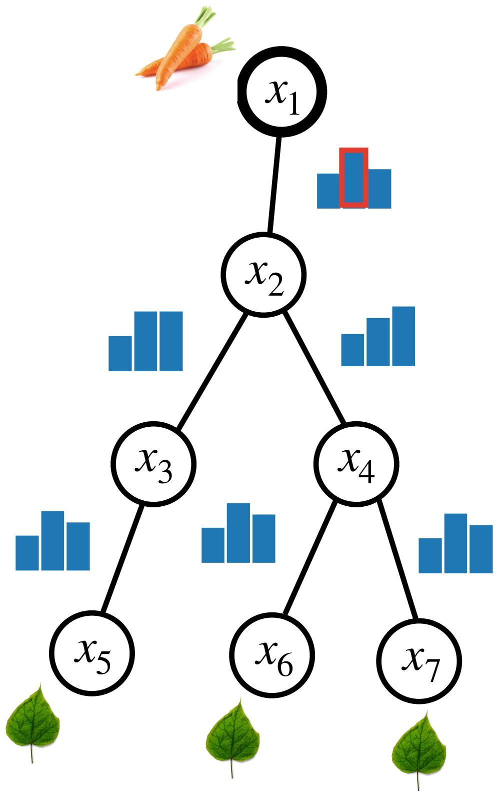
Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \text{argmax}_{x_{\text{root}}} \frac{1}{Z} \prod_{j \sim \text{root}} M_{j \to \text{root}}(x_{\text{root}}).$$



Extension 2. Max-product algorithm

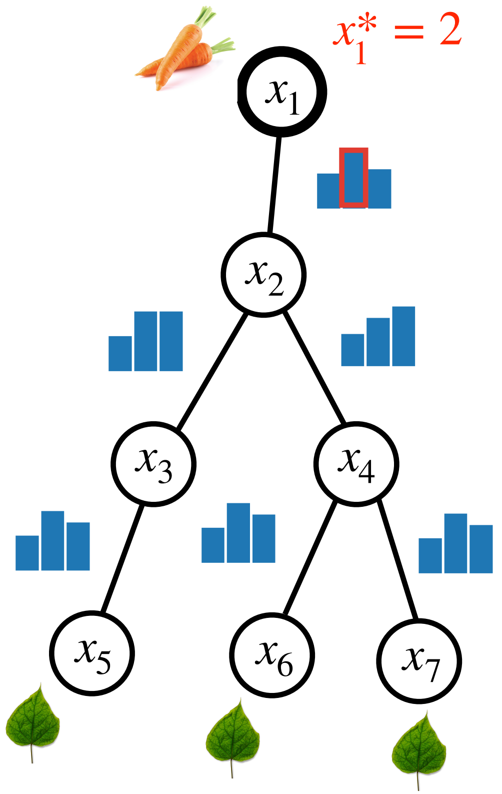
Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \operatorname{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \operatorname{argmax}_{x_{\text{root}}} \frac{1}{Z} \prod_{j \sim \text{root}} M_{j \to \text{root}}(x_{\text{root}}).$$



Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \operatorname{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

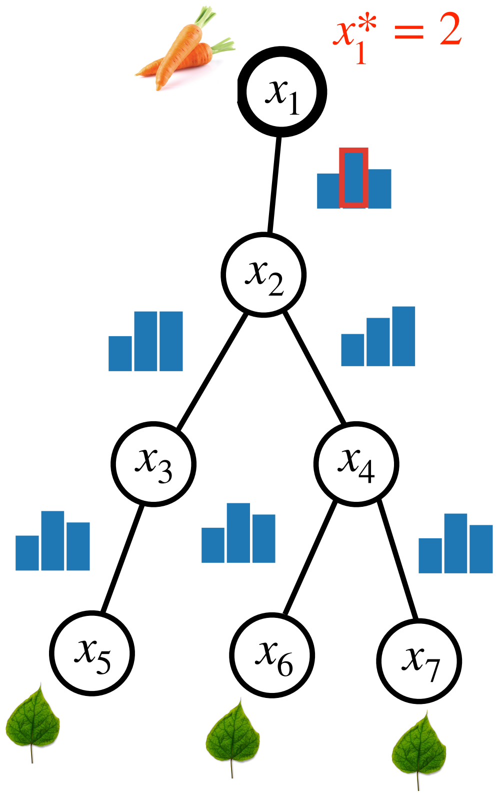
using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \operatorname{argmax}_{x_{\text{root}}} \frac{1}{Z} \prod_{j\sim\text{root}} M_{j\to\text{root}}(x_{\text{root}}).$$

2. From the root node back to the leaf nodes, compute

$$x^*_j = \operatorname{argmax}_{x_j} \psi_{ij}(x^*_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j), \quad j \sim i.$$



Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

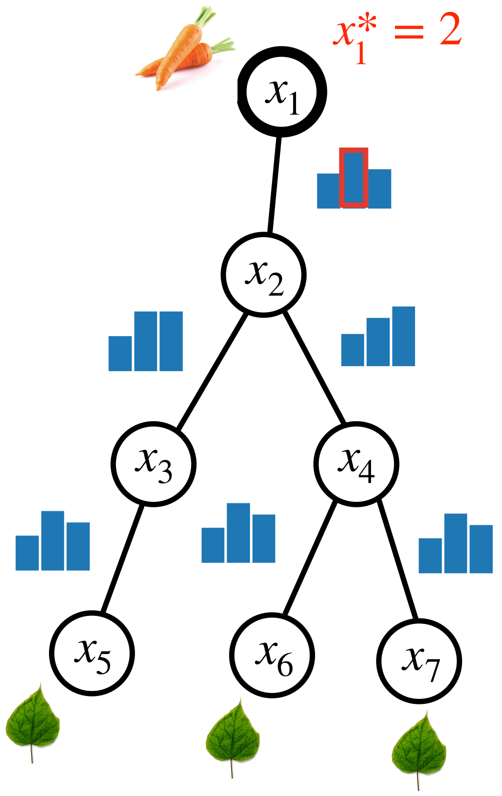
using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \text{argmax}_{x_\text{root}} \frac{1}{Z} \prod_{j \sim \text{root}} M_{j \to \text{root}}(x_\text{root}).$$

2. From the root node back to the leaf nodes, compute

$$x^*_j = \text{argmax}_{x_j} \psi_{ij}(x^*_i, x_j)\, \psi_j(x_j) \prod_{k \sim j, k \neq i} M_{k \to j}(x_j), \quad j \sim i.$$



Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \operatorname{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

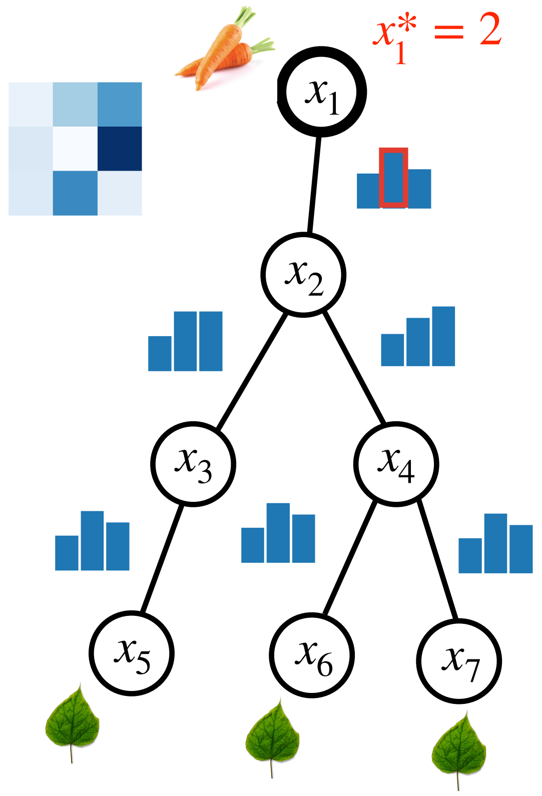
using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \operatorname{argmax}_{x_{\text{root}}} \frac{1}{Z} \prod_{j \sim \text{root}} M_{j \to \text{root}}(x_{\text{root}}).$$

2. From the root node back to the leaf nodes, compute

$$x^*_j = \operatorname{argmax}_{x_j} \psi_{ij}(x^*_i, x_j)\, \psi_j(x_j) \underline{\prod_{k \sim j, k \neq i} M_{k \to j}(x_j)}, \quad j \sim i.$$



Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

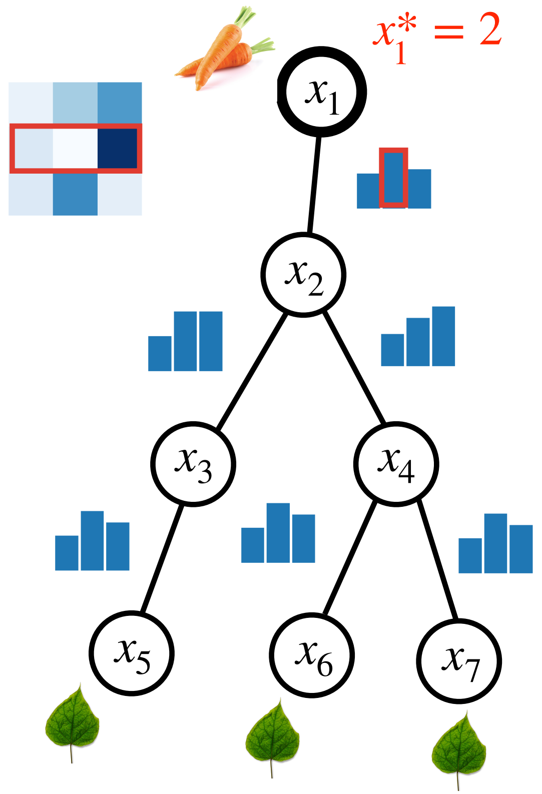
using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \mathrm{argmax}_{x_{\mathrm{root}}} \frac{1}{Z} \prod_{j\sim \mathrm{root}} M_{j\to \mathrm{root}}(x_{\mathrm{root}}).$$

2. From the root node back to the leaf nodes, compute

$$x_j^* = \mathrm{argmax}_{x_j} \psi_{ij}(x_i^*, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j), \quad j \sim i.$$



Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \operatorname{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

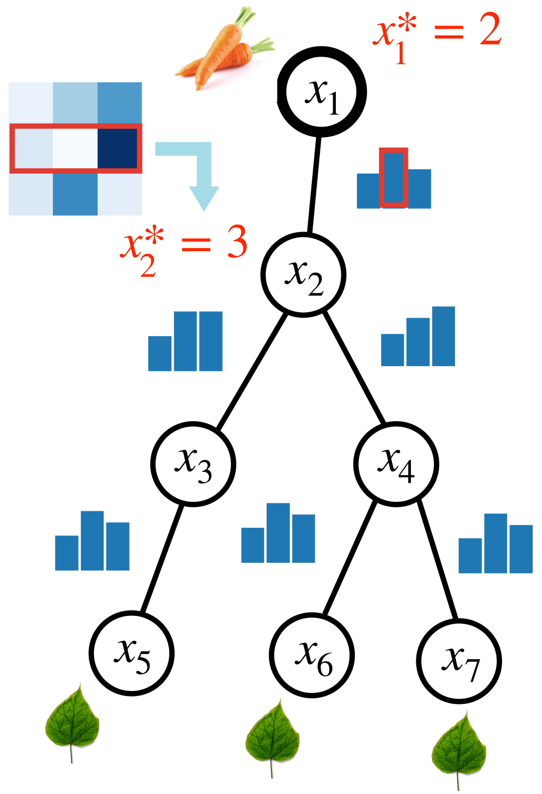
using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \operatorname{argmax}_{x_{\text{root}}} \frac{1}{Z} \prod_{j \sim \text{root}} M_{j \to \text{root}}(x_{\text{root}}).$$

2. From the root node back to the leaf nodes, compute

$$x^*_j = \operatorname{argmax}_{x_j} \psi_{ij}(x^*_i, x_j)\, \psi_j(x_j) \prod_{k \sim j, k \neq i} M_{k \to j}(x_j), \quad j \sim i.$$



Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

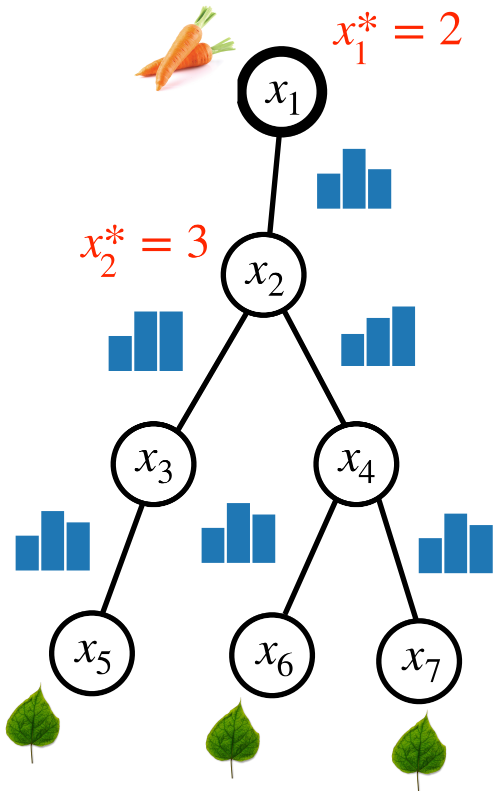
using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \text{argmax}_{x_{\text{root}}} \frac{1}{Z} \prod_{j \sim \text{root}} M_{j \to \text{root}}(x_{\text{root}}).$$

2. From the root node back to the leaf nodes, compute

$$x^*_j = \text{argmax}_{x_j} \psi_{ij}(x^*_i, x_j)\, \psi_j(x_j) \prod_{k \sim j, k \neq i} M_{k \to j}(x_j), \quad j \sim i.$$



Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

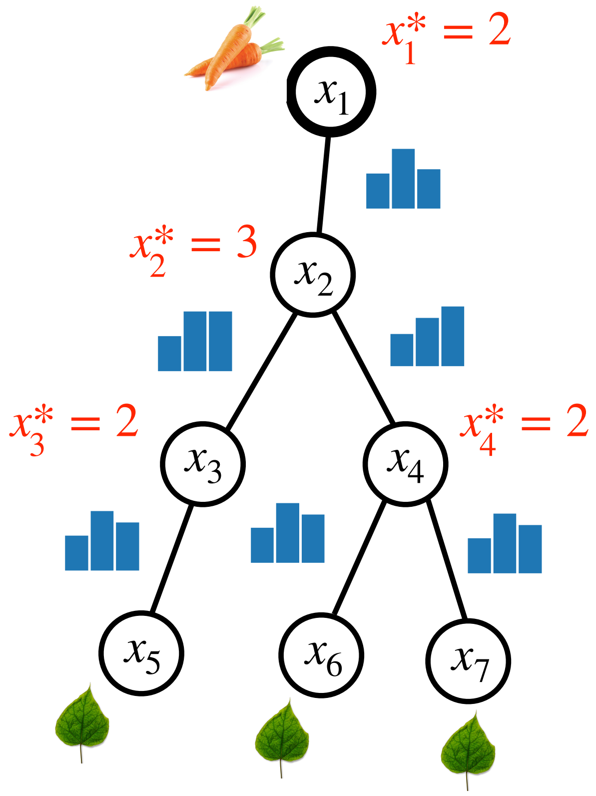
using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \mathrm{argmax}_{x_{\mathrm{root}}} \frac{1}{Z} \prod_{j\sim \mathrm{root}} M_{j\to \mathrm{root}}(x_{\mathrm{root}}).$$

2. From the root node back to the leaf nodes, compute

$$x^*_j = \mathrm{argmax}_{x_j} \psi_{ij}(x^*_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j), \quad j \sim i.$$



Extension 2. Max-product algorithm

Going from the root node back to the leaf nodes, we can find the **mode**:

$$\boldsymbol{x}^* = \mathrm{argmax}_{\boldsymbol{x}} p(\boldsymbol{x}),$$

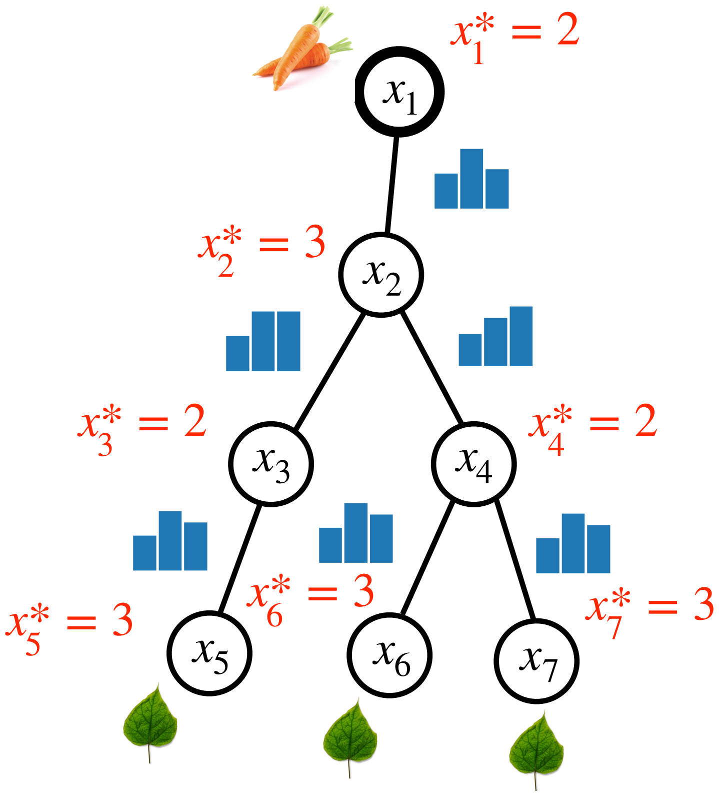
using a procedure called *back-tracking:*

1. At the root node, compute

$$x^*_{root} = \mathrm{argmax}_{x_{\mathrm{root}}} \frac{1}{Z} \prod_{j \sim \mathrm{root}} M_{j \to \mathrm{root}}(x_{\mathrm{root}}).$$

2. From the root node back to the leaf nodes, compute

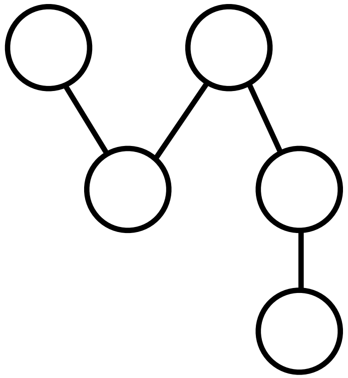
$$x^*_j = \mathrm{argmax}_{x_j} \psi_{ij}(x^*_i, x_j)\, \psi_j(x_j) \prod_{k \sim j, k \neq i} M_{k \to j}(x_j), \quad j \sim i.$$



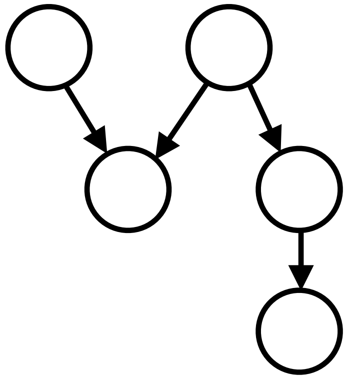
Extension 3. Polytrees and other graphs

Extension 3. Polytrees and other graphs

A **polytree** is a directed tree



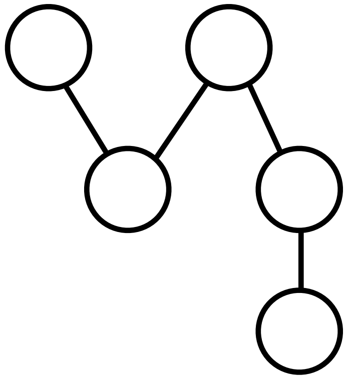
A tree



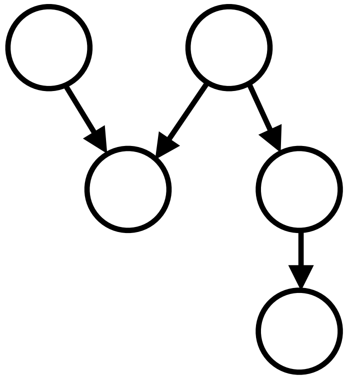
A polytree

Extension 3. Polytrees and other graphs

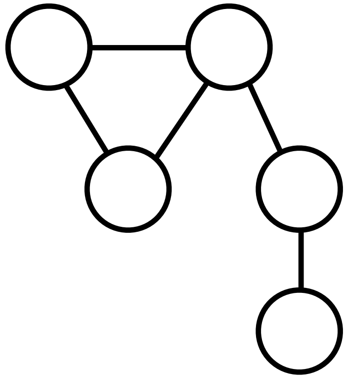
A **polytree** is a directed tree



A tree



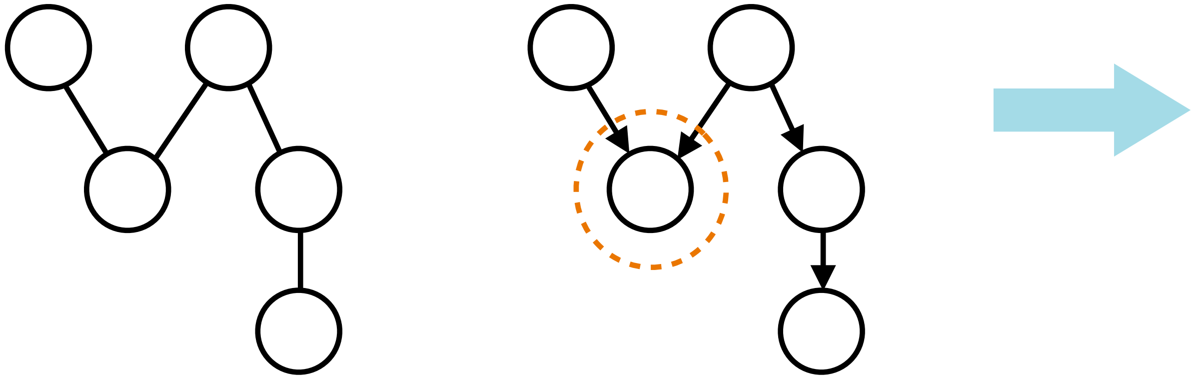
A polytree



A polytree as a MRF

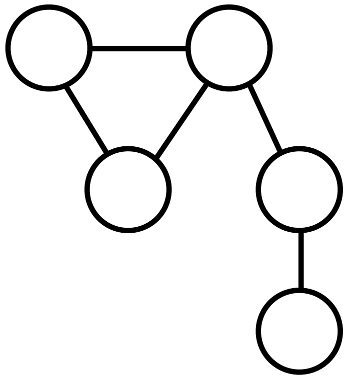
Extension 3. Polytrees and other graphs

A **polytree** is a directed tree



A tree

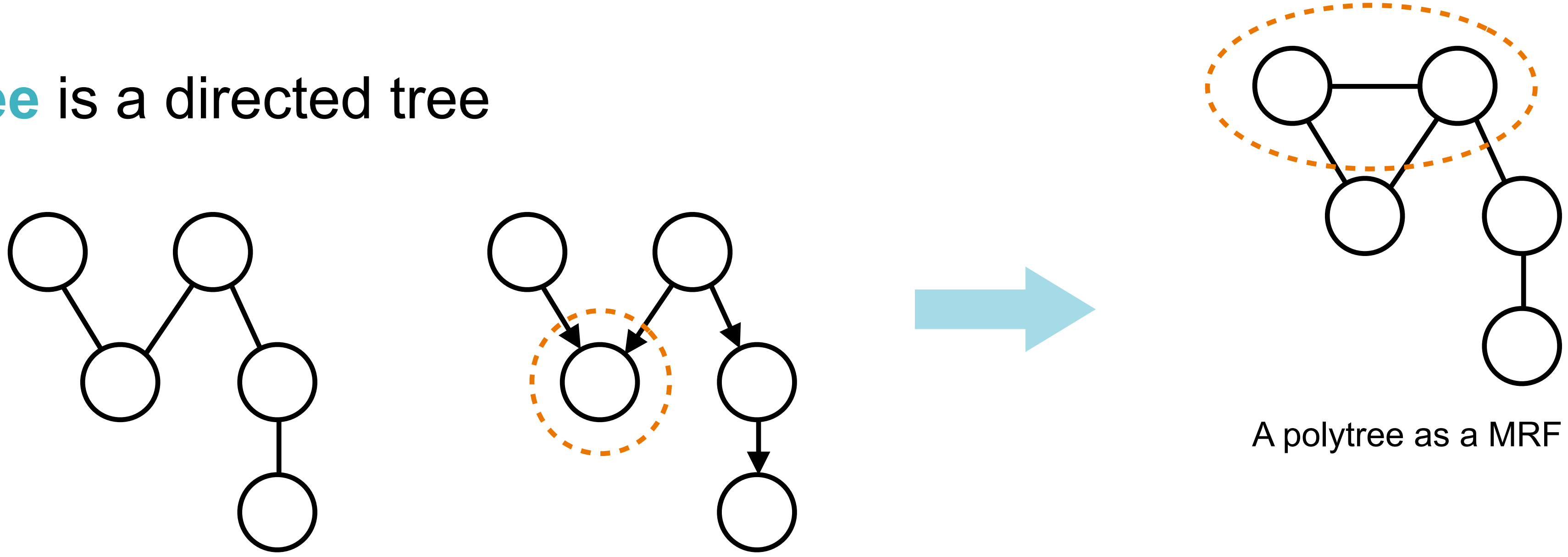
A polytree



A polytree as a MRF

Extension 3. Polytrees and other graphs

A **polytree** is a directed tree



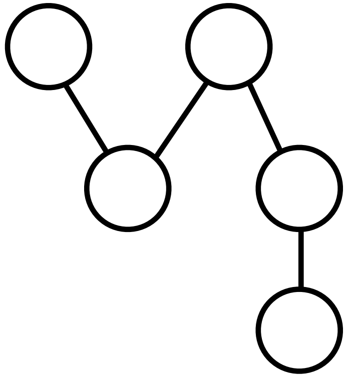
A tree

A polytree

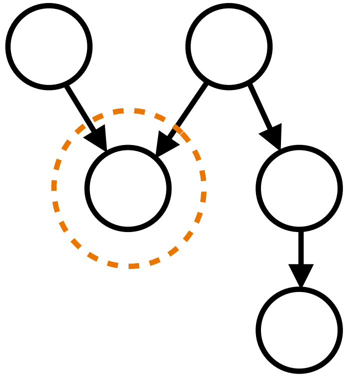
A polytree as a MRF

Extension 3. Polytrees and other graphs

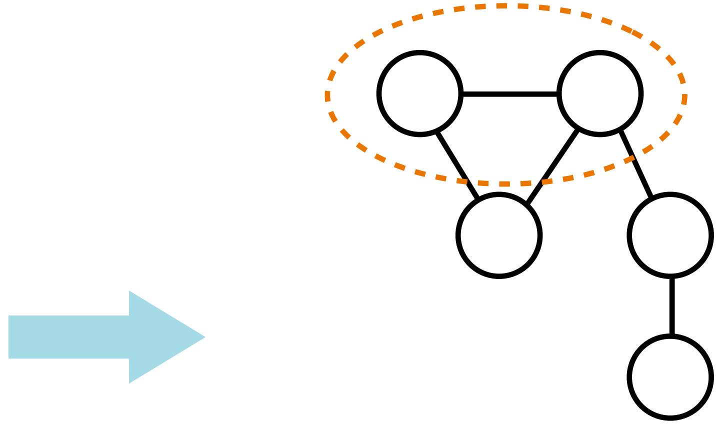
A **polytree** is a directed tree



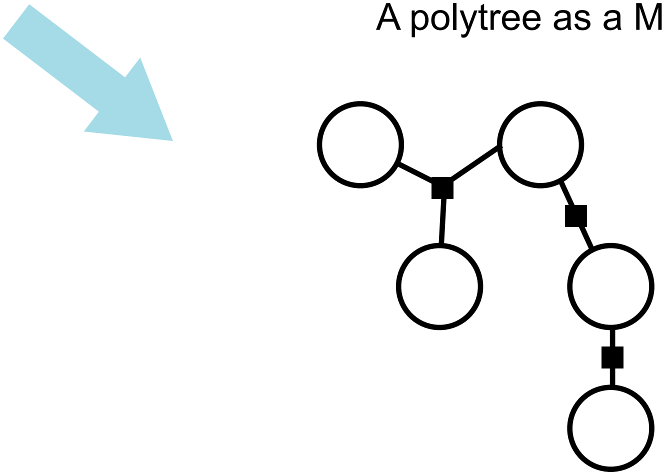
A tree



A polytree



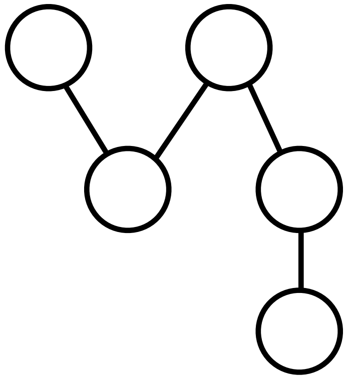
A polytree as a MRF



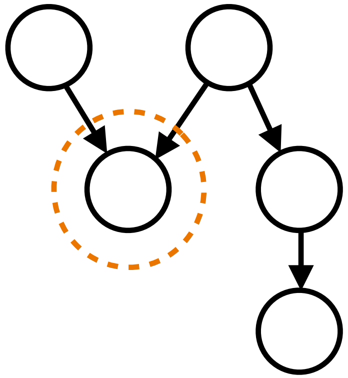
A polytree as a factor graph

Extension 3. Polytrees and other graphs

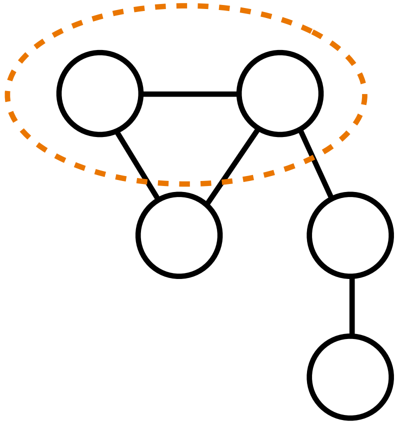
A **polytree** is a directed tree



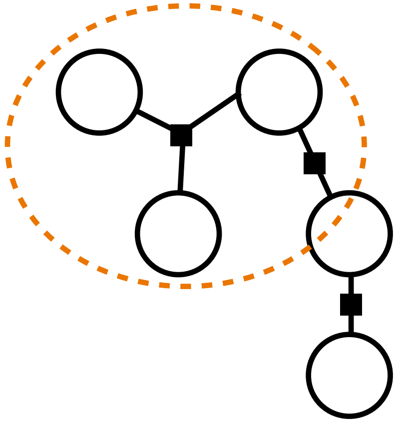
A tree



A polytree



A polytree as a MRF



A polytree as a factor graph

Note: factors are not necessarily pairwise!

Extension 3. Polytrees and other graphs

On trees, the message passing updates read:

Message update:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

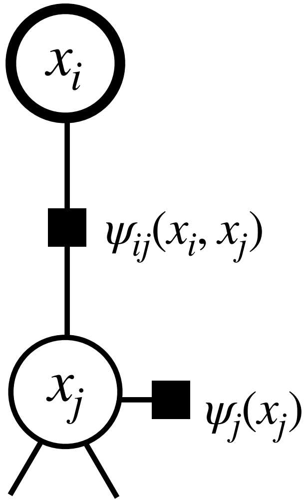
State update:

$$p(x_i) \propto \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$

Extension 3. Polytrees and other graphs

First, break down the message update step into two sub-steps:

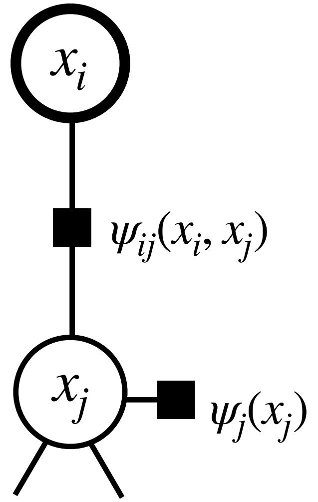
$$M_{j\to i}(x_i) = \sum_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$



Extension 3. Polytrees and other graphs

First, break down the message update step into two sub-steps:

$$\boxed{M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k \sim j, k \neq i} M_{k\to j}(x_j),}$$

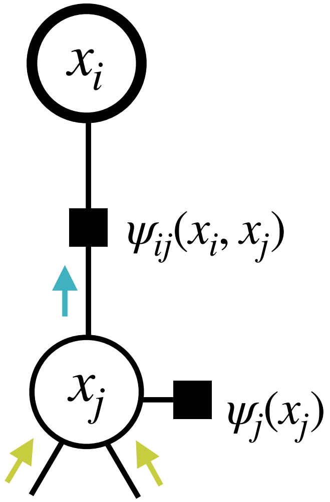


Extension 3. Polytrees and other graphs

First, break down the message update step into two sub-steps:

$$M_{j\to i}(x_i) = \sum_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

1. $\mu_{x_j\to\psi_{ij}}(x_j) = \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j).$ (variable-to-factor message)



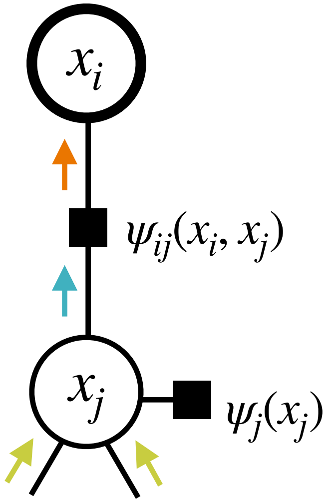
Extension 3. Polytrees and other graphs

First, break down the message update step into two sub-steps:

$$\boxed{M_{j\to i}(x_i) = \sum_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),}$$

1. $\mu_{x_j\to\psi_{ij}}(x_j) = \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j).$ (variable-to-factor message)

2. $\mu_{\psi_{ij}\to x_i}(x_i) = \sum_{x_j\in\{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \mu_{x_j\to\psi_{ij}}(x_j).$ (factor-to-variable message)



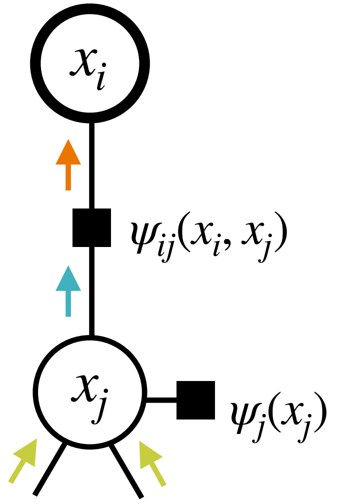
Extension 3. Polytrees and other graphs

First, break down the message update step into two sub-steps:

$$\boxed{M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),}$$

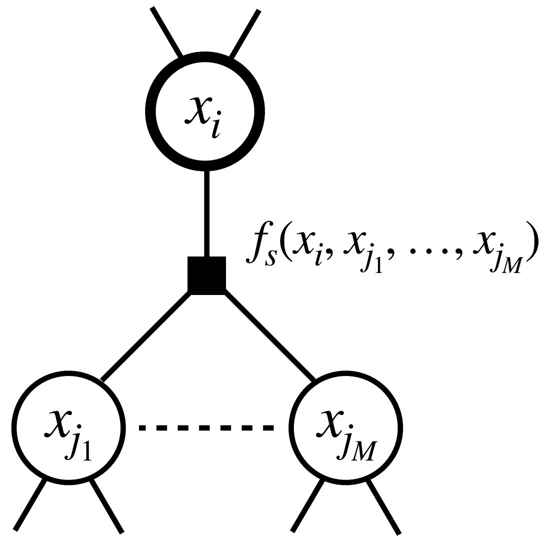
1. $\mu_{x_j \to \psi_{ij}}(x_j) = \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j).$ (variable-to-factor message)

2. $\underbrace{\mu_{\psi_{ij} \to x_i}(x_i)}_{\equiv M_{j\to i}(x_i)} = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \mu_{x_j \to \psi_{ij}}(x_j).$ (factor-to-variable message)



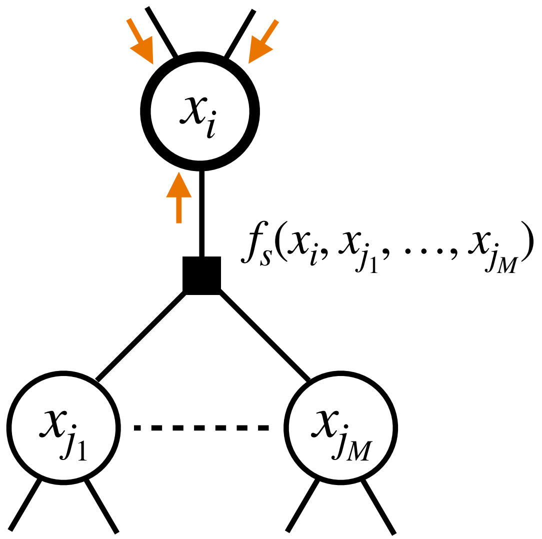
Extension 3. Polytrees and other graphs

Extending to polytrees:



Extension 3. Polytrees and other graphs

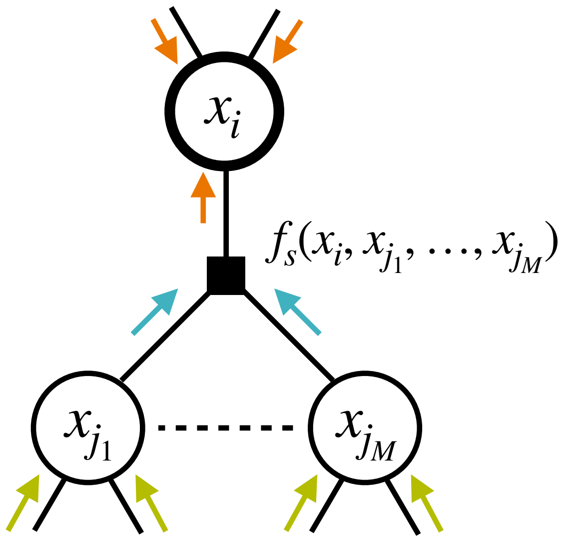
Extending to polytrees:



Extension 3. Polytrees and other graphs

Extending to polytrees:

1. $\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \setminus s} \mu_{f_l \to x_j}(x_j)$

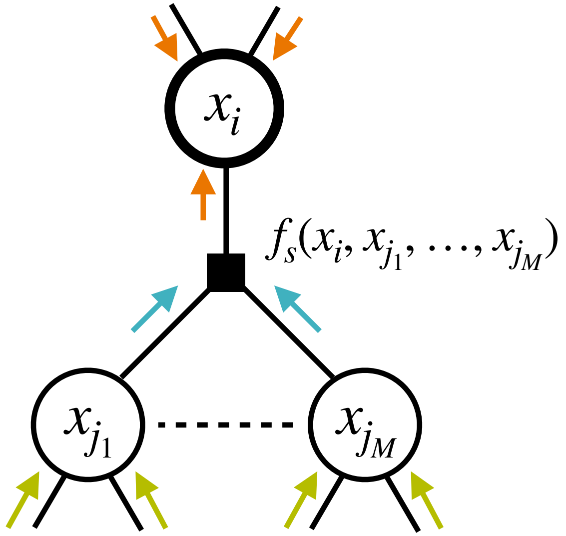


Extension 3. Polytrees and other graphs

Extending to polytrees:

1. $\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \setminus s} \mu_{f_l \to x_j}(x_j)$

2. $\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \ldots, x_{j_M}} f_s(x_i, x_{j_1}, \ldots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})$

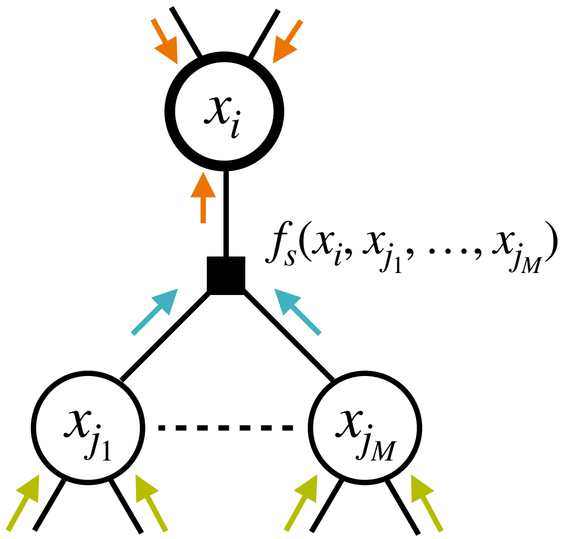


Extension 3. Polytrees and other graphs

Extending to polytrees:

1. $\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \setminus s} \mu_{f_l \to x_j}(x_j)$

2. $\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \ldots, x_{j_M}} f_s(x_i, x_{j_1}, \ldots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})$



The state updates read:

$$p(x_i) = \prod_{s \in \mathrm{ne}(x_i)} \mu_{f_s \to x_i}(x_i).$$

Extension 3. Polytrees and other graphs

We can apply the same update rules to more general graphs with loops.

This is called **Loopy Belief Propagation (LBP)**.

Extension 3. Polytrees and other graphs

We can apply the same update rules to more general graphs with loops.

This is called **Loopy Belief Propagation (LBP)**.

Message update (same as before):

1. $\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \backslash s} \mu_{f_l \to x_j}(x_j)$

2. $\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \ldots, x_{j_M}} f_s(x_i, x_{j_1}, \ldots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})$

State update (same as before):

$$p(x_i) = \prod_{s \in \mathrm{ne}(x_i)} \mu_{f_s \to x_i}(x_i).$$

Extension 3. Polytrees and other graphs

We can apply the same update rules to more general graphs with loops.

This is called **Loopy Belief Propagation (LBP)**.

Message update (same as before):

1. $\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \setminus s} \mu_{f_l \to x_j}(x_j)$

2. $\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \ldots, x_{j_M}} f_s(x_i, x_{j_1}, \ldots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})$

State update (same as before):

$$p(x_i) = \prod_{s \in \mathrm{ne}(x_i)} \mu_{f_s \to x_i}(x_i).$$

- LBP is iterative and can be started off by setting

$$\mu_{x \to f}(x) = 1,$$

for all variables x and factors f.

Extension 3. Polytrees and other graphs

We can apply the same update rules to more general graphs with loops.

This is called **Loopy Belief Propagation (LBP)**.

Message update (same as before):

1. $\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \backslash s} \mu_{f_l \to x_j}(x_j)$

2. $\boxed{\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \ldots, x_{j_M}} f_s(x_i, x_{j_1}, \ldots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})}$

State update (same as before):

$$p(x_i) = \prod_{s \in \mathrm{ne}(x_i)} \mu_{f_s \to x_i}(x_i).$$

- LBP is iterative and can be started off by setting

$$\mu_{x \to f}(x) = 1,$$

for all variables x and factors f.

Extension 3. Polytrees and other graphs

We can apply the same update rules to more general graphs with loops.

This is called **Loopy Belief Propagation (LBP)**.

Message update (same as before):

1. $\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \setminus s} \mu_{f_l \to x_j}(x_j)$

2. $\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \dots, x_{j_M}} f_s(x_i, x_{j_1}, \dots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})$

State update (same as before):

$$p(x_i) = \prod_{s \in \mathrm{ne}(x_i)} \mu_{f_s \to x_i}(x_i).$$

- LBP is iterative and can be started off by setting

$$\mu_{x \to f}(x) = 1,$$

for all variables x and factors f.

Extension 3. Polytrees and other graphs

We can apply the same update rules to more general graphs with loops.

This is called **Loopy Belief Propagation (LBP)**.

Message update (same as before):

1. $\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \setminus s} \mu_{f_l \to x_j}(x_j)$

2. $\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \ldots, x_{j_M}} f_s(x_i, x_{j_1}, \ldots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})$

State update (same as before):

$$p(x_i) = \prod_{s \in \mathrm{ne}(x_i)} \mu_{f_s \to x_i}(x_i).$$

- LBP is iterative and can be started off by setting

$$\mu_{x \to f}(x) = 1,$$

for all variables x and factors f.

Extension 3. Polytrees and other graphs

We can apply the same update rules to more general graphs with loops.

This is called **Loopy Belief Propagation (LBP)**.

Message update (same as before):

1. $$\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \setminus s} \mu_{f_l \to x_j}(x_j)$$

2. $$\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \ldots, x_{j_M}} f_s(x_i, x_{j_1}, \ldots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})$$

State update (same as before):

$$p(x_i) = \prod_{s \in \mathrm{ne}(x_i)} \mu_{f_s \to x_i}(x_i).$$

- LBP is iterative and can be started off by setting

$$\mu_{x \to f}(x) = 1,$$

for all variables x and factors f.

Extension 3. Polytrees and other graphs

We can apply the same update rules to more general graphs with loops.

This is called **Loopy Belief Propagation (LBP)**.

Message update (same as before):

1. $$\mu_{x_j \to f_s}(x_j) = \prod_{l \in \mathrm{ne}(x_j) \setminus s} \mu_{f_l \to x_j}(x_j)$$

2. $$\mu_{f_s \to x_i}(x_i) = \sum_{x_{j_1}, \ldots, x_{j_M}} f_s(x_i, x_{j_1}, \ldots, x_{j_M}) \prod_{k=1}^{M} \mu_{x_{j_k} \to f_s}(x_{j_k})$$

State update (same as before):

$$p(x_i) = \prod_{s \in \mathrm{ne}(x_i)} \mu_{f_s \to x_i}(x_i).$$

- LBP is iterative and can be started off by setting

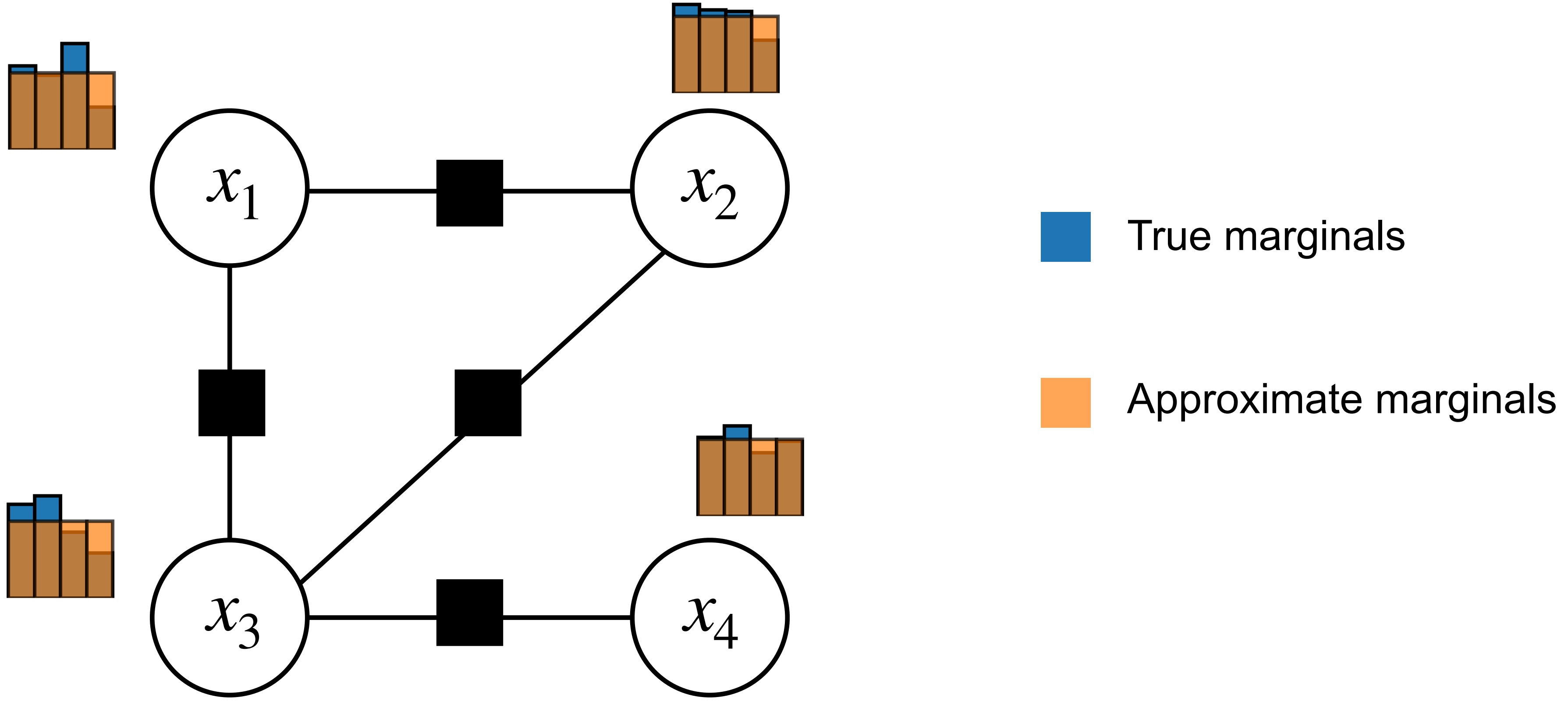
  $$\mu_{x \to f}(x) = 1,$$

  for all variables x and factors f.

- Updates can be done in parallel (flooding schedule).

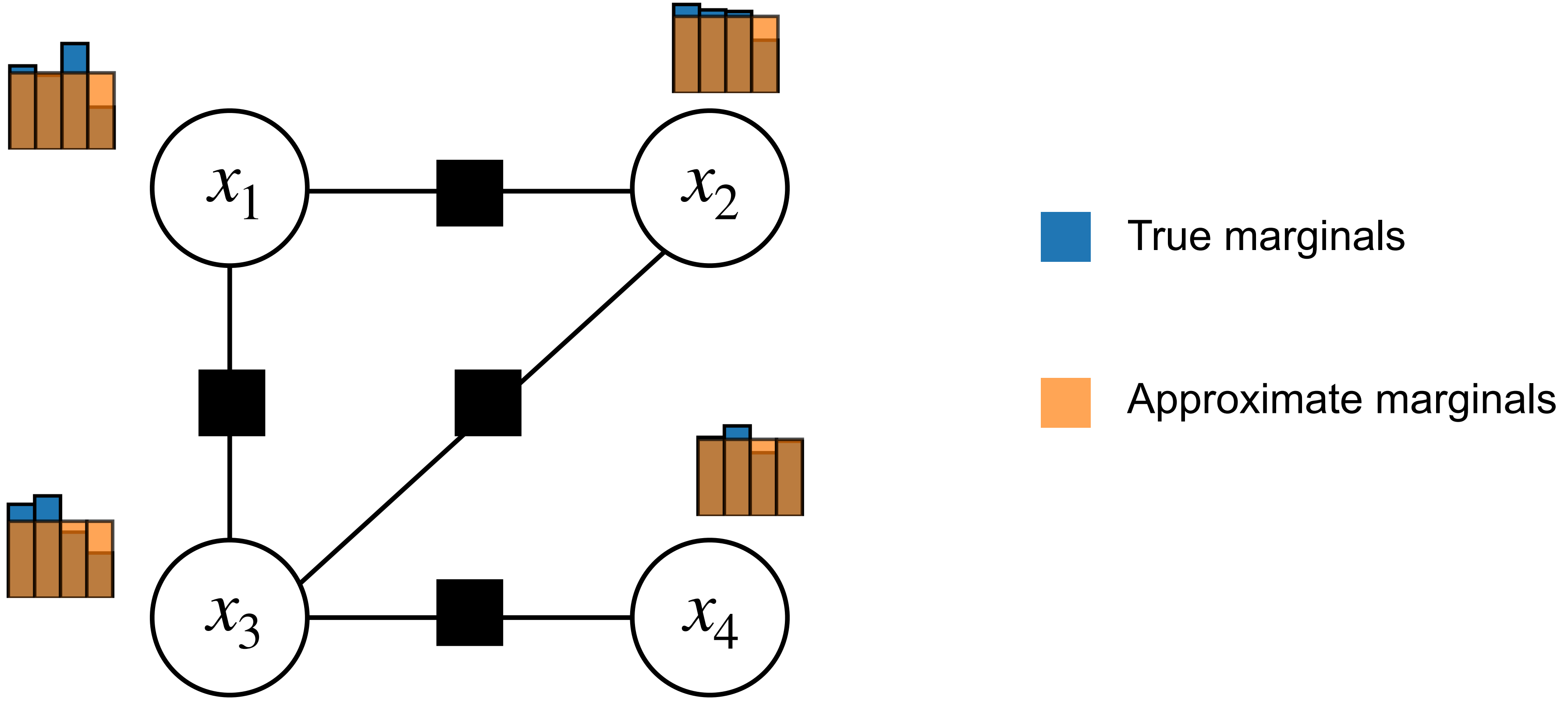
Implementation (flooding schedule)

Iteration 1



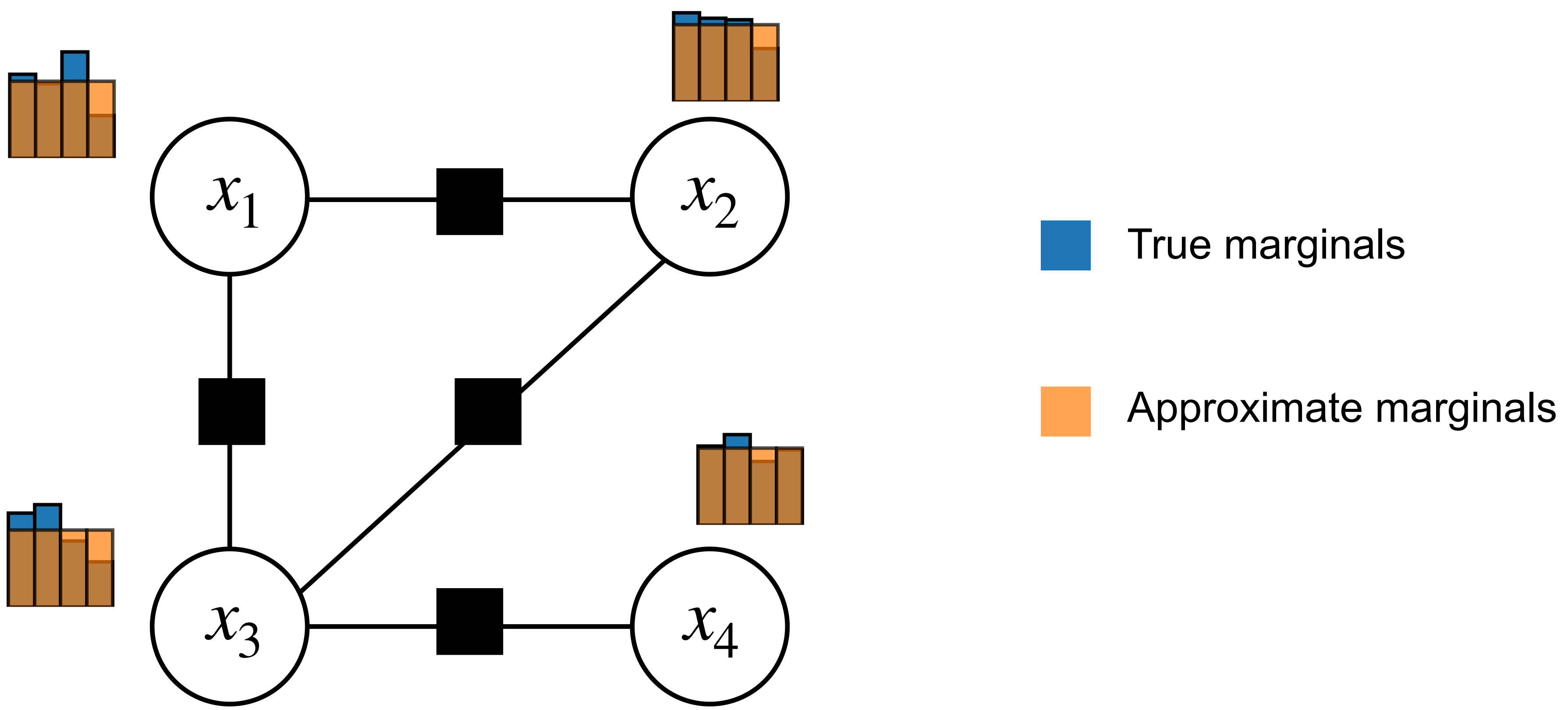
Implementation (flooding schedule)

Iteration 1



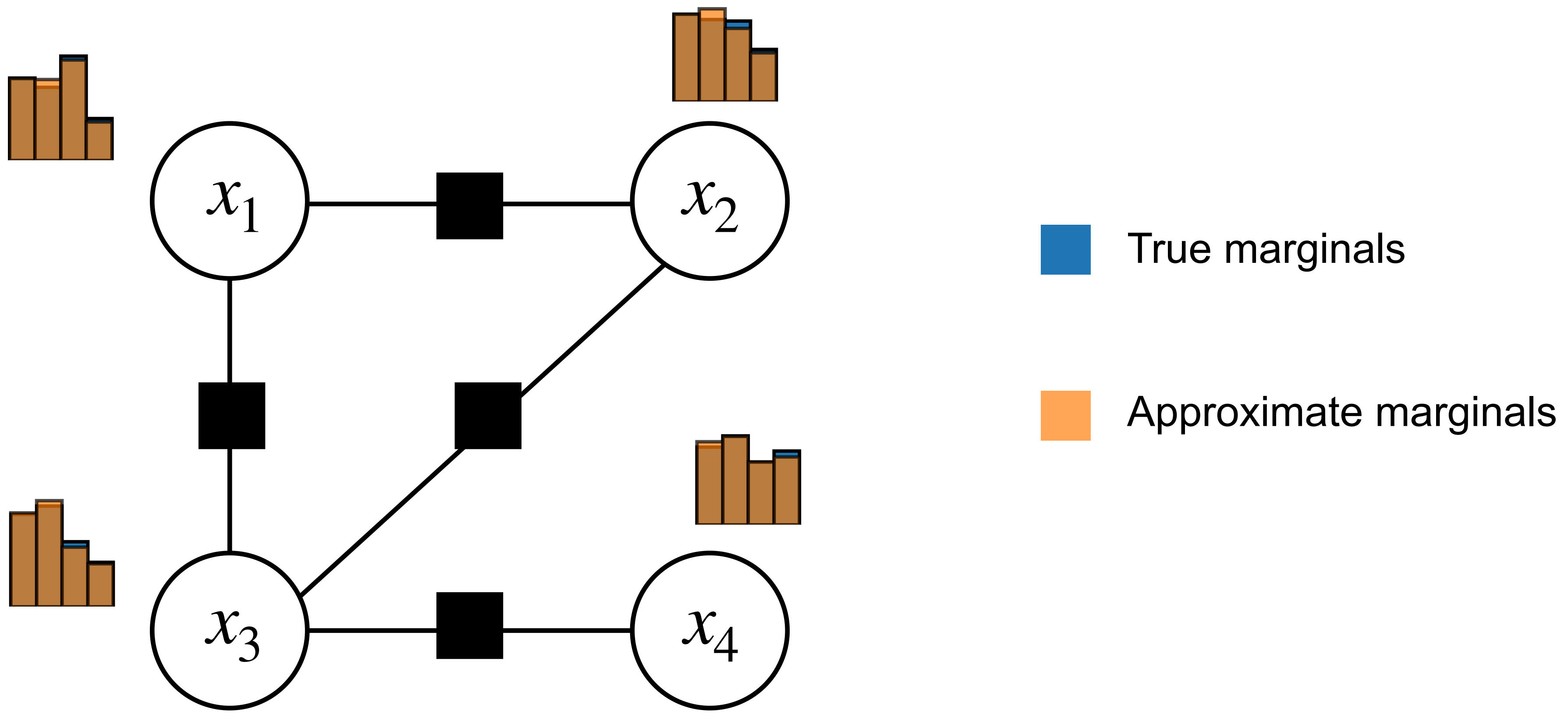
Implementation (flooding schedule)

Iteration 1



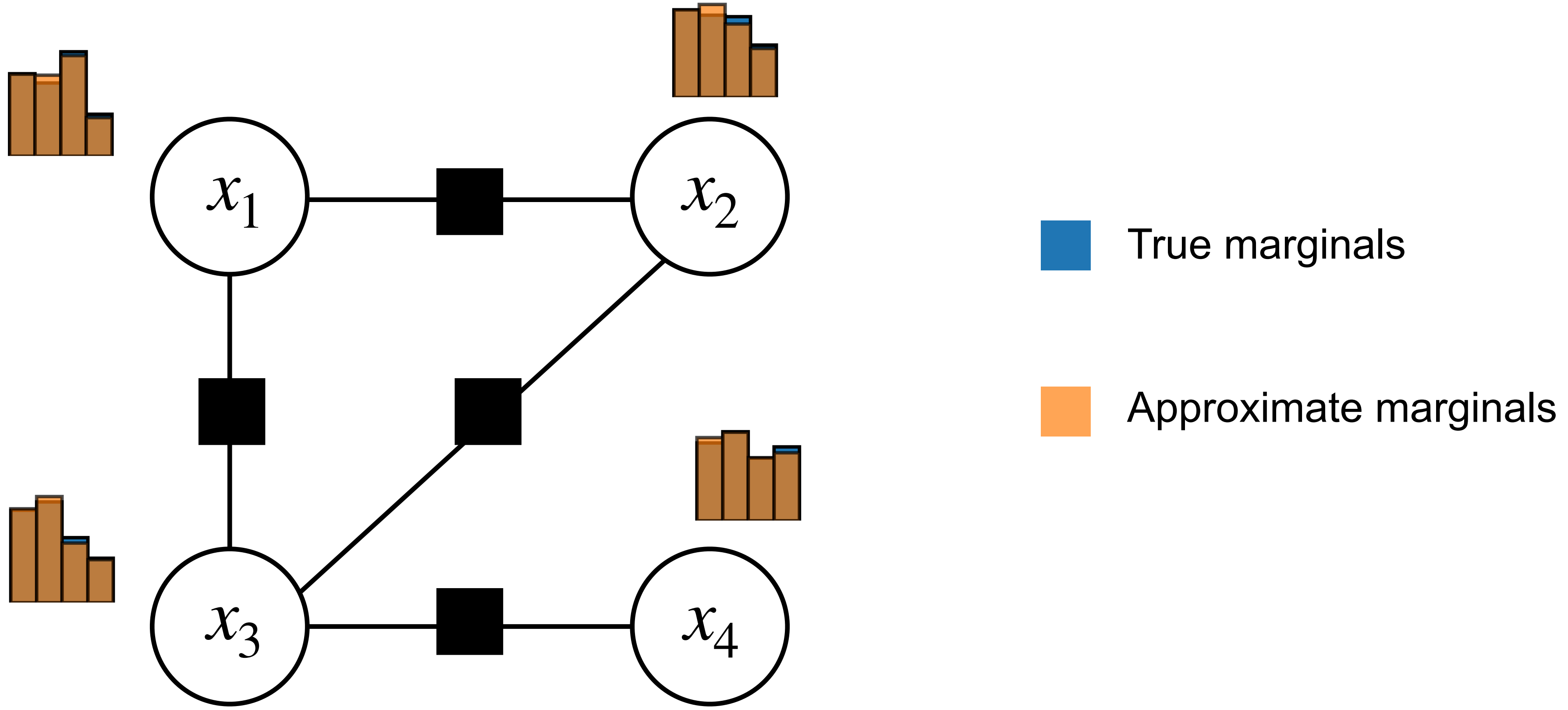
Implementation (flooding schedule)

Iteration 1



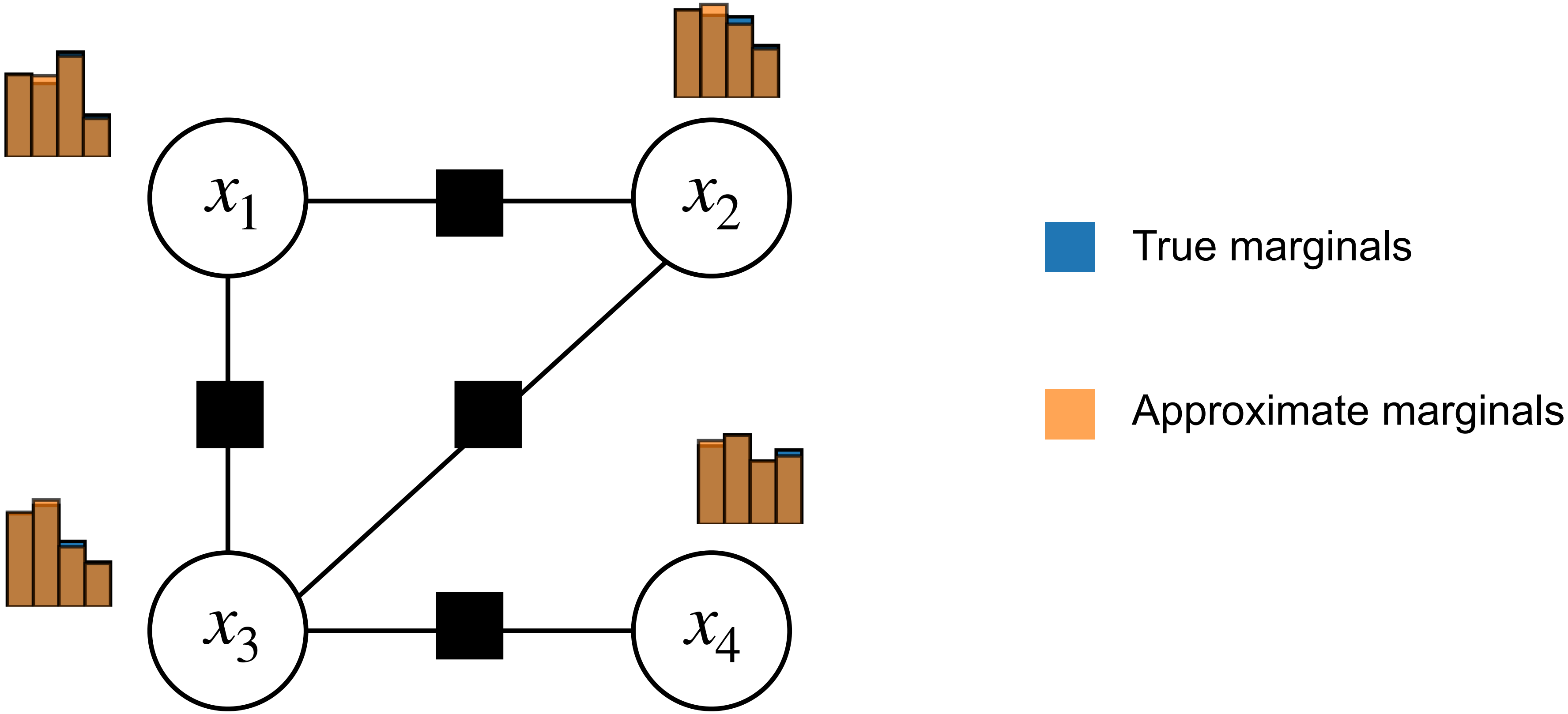
Implementation (flooding schedule)

Iteration 1



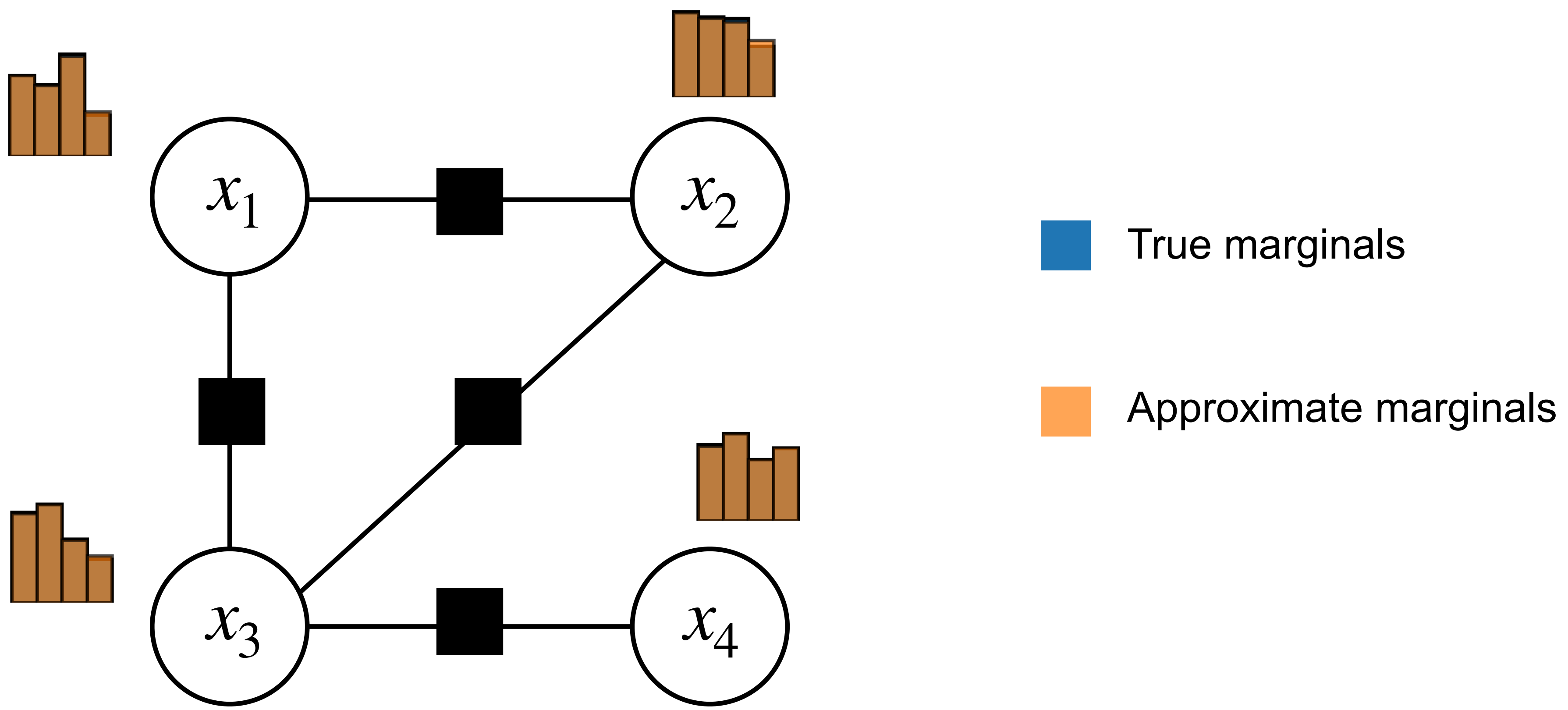
Implementation (flooding schedule)

Iteration 1



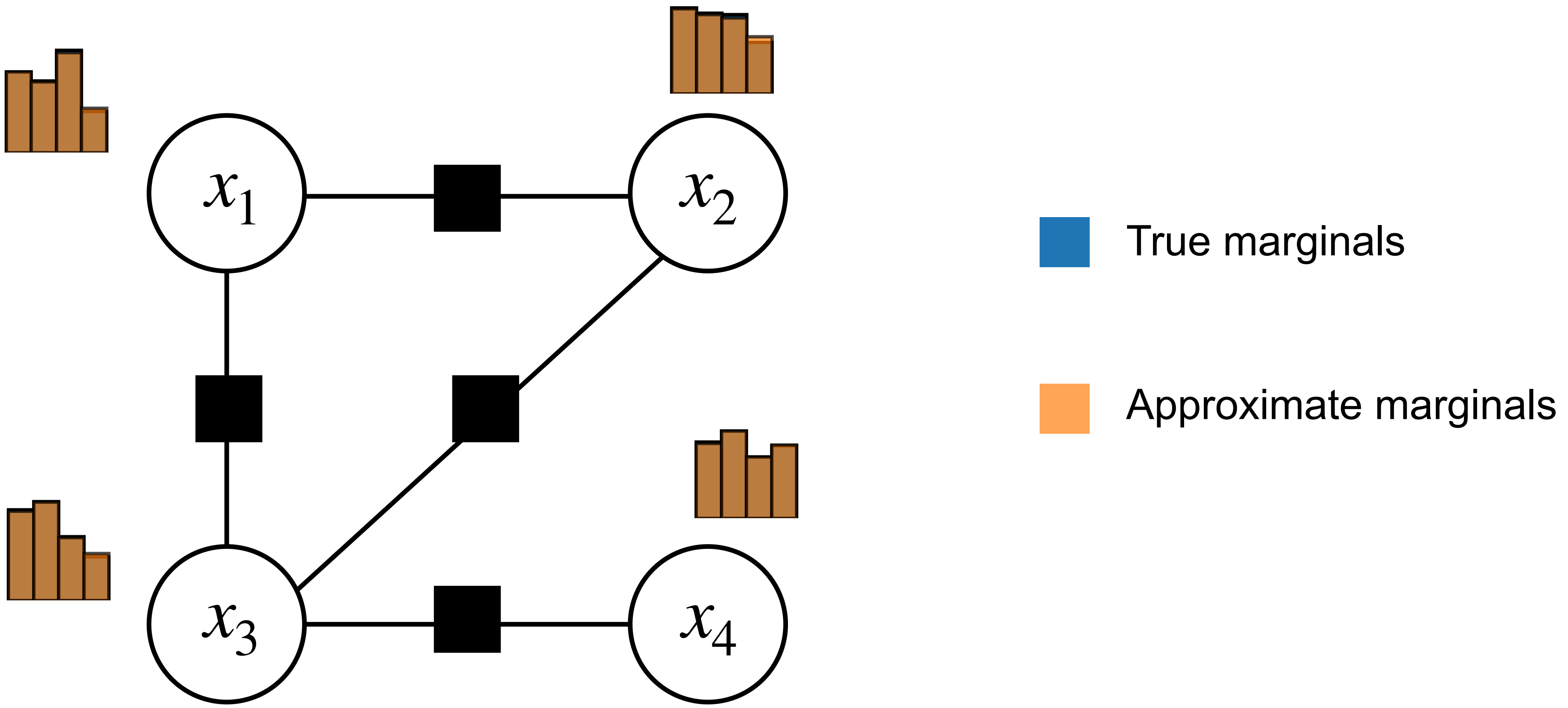
Implementation (flooding schedule)

Iteration 2



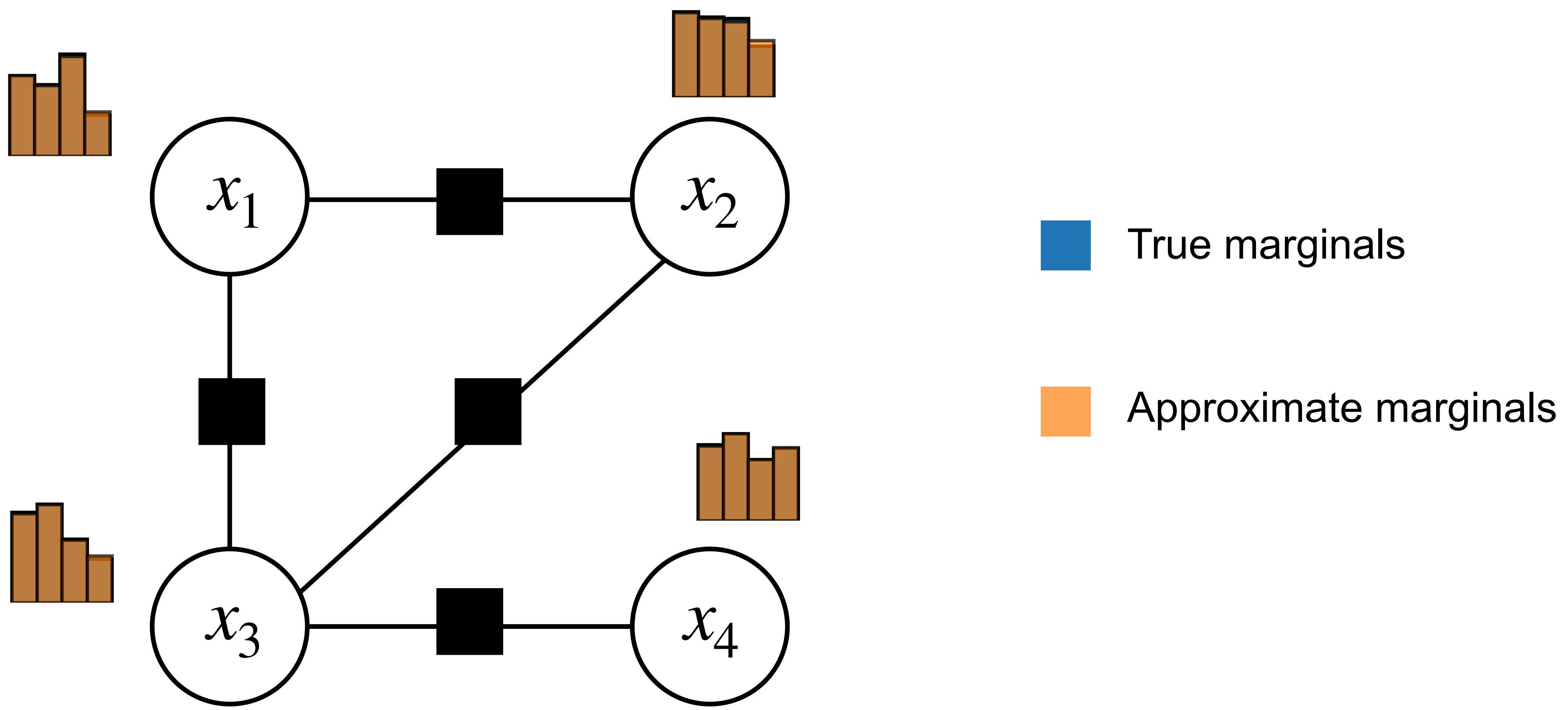
Implementation (flooding schedule)

Iteration 2



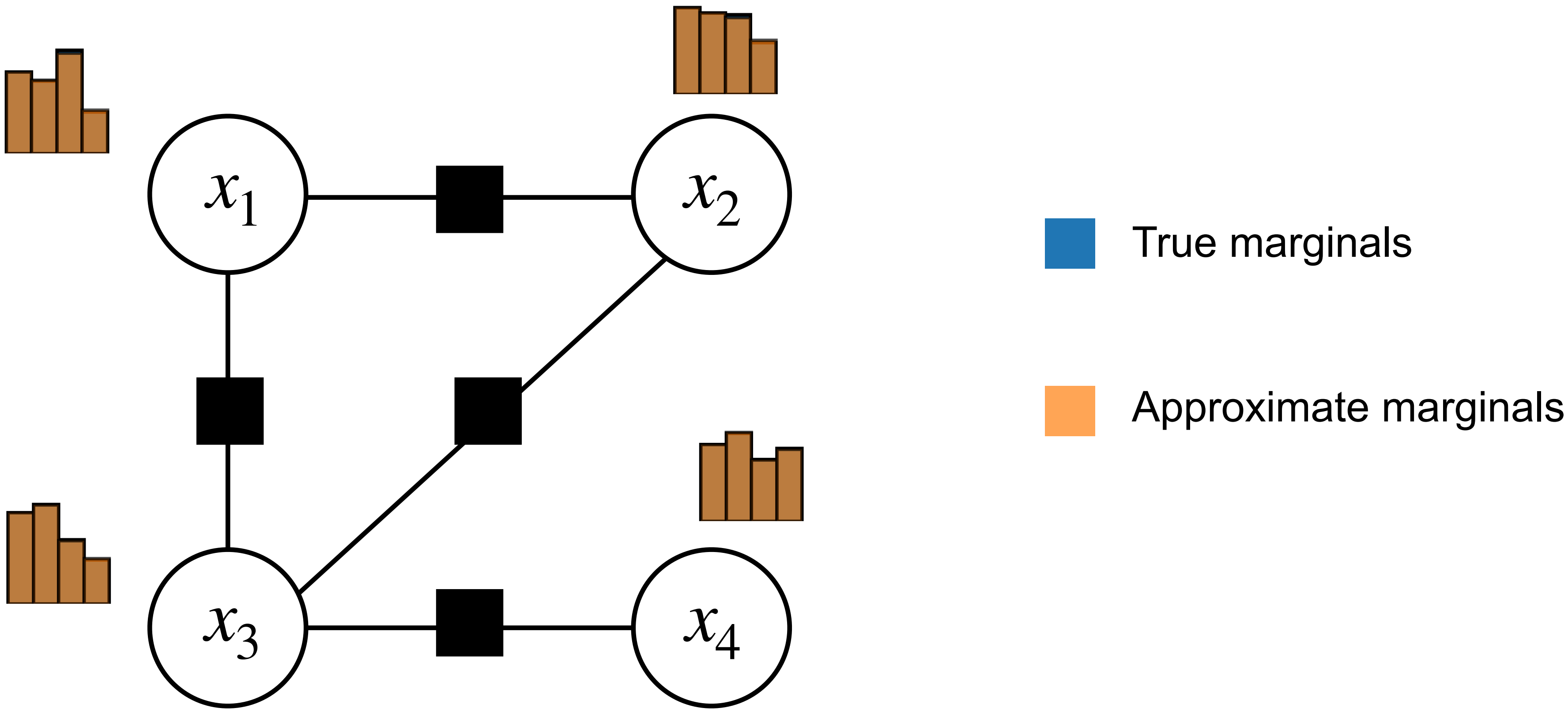
Implementation (flooding schedule)

Iteration 2



Implementation (flooding schedule)

Iteration 3



Remarks

Remarks

- LBP *does not* have any convergence guarantee

Remarks

- LBP *does not* have any convergence guarantee
- But when it converges, the results are usually good

Remarks

- LBP *does not* have any convergence guarantee
- But when it converges, the results are usually good
- On trees/polytrees, convergence is guaranteed

Remarks

- LBP *does not* have any convergence guarantee
- But when it converges, the results are usually good
- On trees/polytrees, convergence is guaranteed
- Some variations of LBP exists, most notably **expectation propagation** [4]**:**
  - Approximates intractable distributions by a product of simpler ones
  - Closeness is measured by the Kullback-Leibler (KL) divergence
  - When applied to graphs, it generalises LBP [4]

Remarks

- LBP *does not* have any convergence guarantee
- But when it converges, the results are usually good
- On trees/polytrees, convergence is guaranteed
- Some variations of LBP exists, most notably **expectation propagation** [4]**:**
  - Approximates intractable distributions by a product of simpler ones
  - Closeness is measured by the Kullback-Leibler (KL) divergence
  - When applied to graphs, it generalises LBP [4]
- LBP is closely related to Bethe free energy optimisation [5]

References

[1] Bishop, Christopher M. *Pattern Recognition and Machine Learning*. New York: springer, 2006.

[2] Wainwright, Martin J., and Michael I. Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Foundations and Trends in Machine Learning, 2008.

[3] Ortiz, Joseph, Talfan Evans, and Andrew J. Davison. *A Visual Introduction to Gaussian Belief Propagation.* 2021. (https://gaussianbp.github.io/)

[4] Minka, Thomas P. *Expectation propagation for approximate Bayesian inference*. Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence, 2001.

[5] Yedidia, Jonathan S., William T. Freeman, and Yair Weiss. *Understanding belief propagation and its generalizations.* Exploring artificial intelligence in the new millennium, 2003.

4. Message Passing Neural Networks

Neural networks

Neural networks have dominated ML in the past decade.

Neural networks

Neural networks have dominated ML in the past decade.

They are:

Neural networks

Neural networks have dominated ML in the past decade.

They are:

- Extremely flexible for modelling

Neural networks

Neural networks have dominated ML in the past decade.

They are:

- Extremely flexible for modelling
- Able to process complex data structures

Neural networks

Neural networks have dominated ML in the past decade.

They are:

- Extremely flexible for modelling
- Able to process complex data structures
- Composed of simple, parallelisable components

Neural networks

Neural networks have dominated ML in the past decade.

They are:

- Extremely flexible for modelling
- Able to process complex data structures
- Composed of simple, parallelisable components
- Automatically differentiable

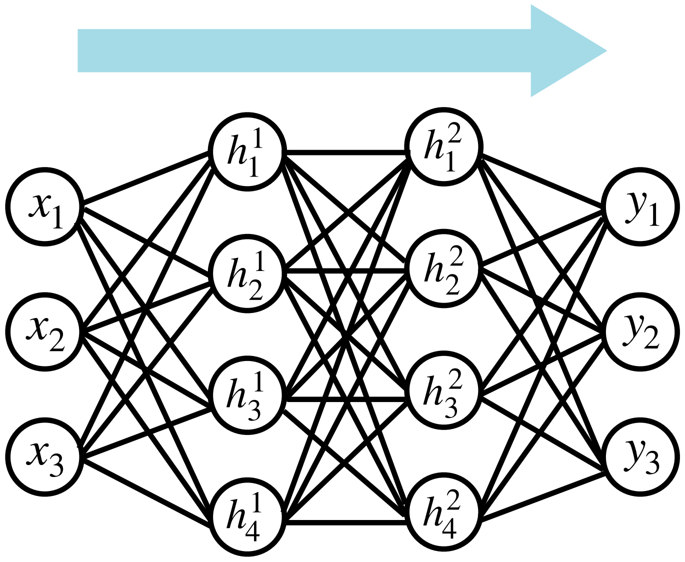
Neural networks

Neural networks have dominated ML in the past decade.

They are:

- Extremely flexible for modelling
- Able to process complex data structures
- Composed of simple, parallelisable components
- Automatically differentiable

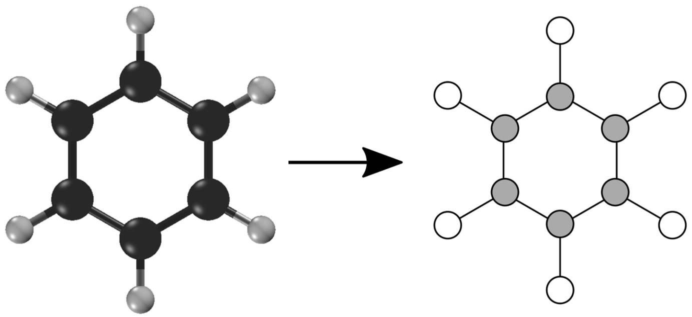
$$h^0 = x$$
$$h^{l+1} = \mathrm{ReLU}(Wh^l + b), \quad t = 0, \ldots, L-1$$
$$y = \mathrm{Softmax}(Wh^L + b)$$



Multilayer perceptron

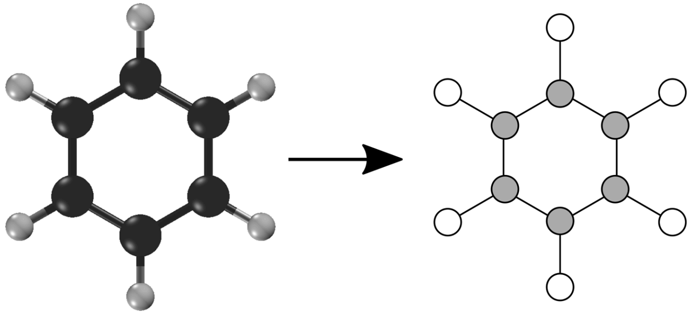
A zoo of graphs in the real world

A zoo of graphs in the real world



Molecules as graphs
Image from: https://www.oreilly.com/library/view/deep-learning-for/9781492039822/ch04.html

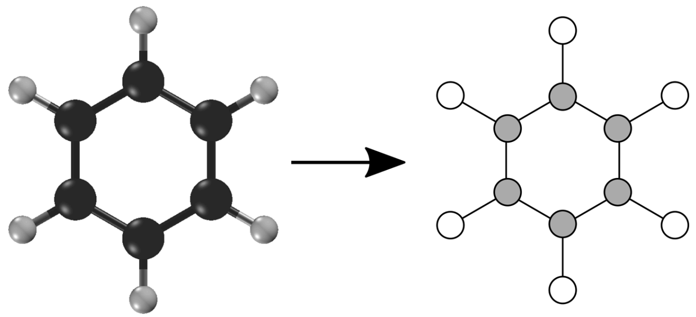
A zoo of graphs in the real world



Molecules as graphs
Image from: https://www.oreilly.com/library/view/deep-learning-for/9781492039822/ch04.html

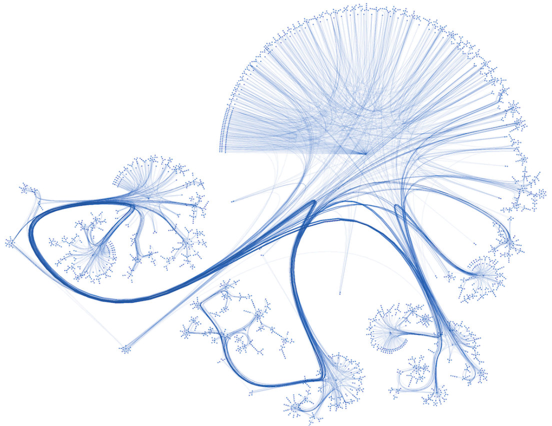
Social networks
Image from: https://medium.com/analytics-vidhya/social-network-analytics-f082f4e21b16

A zoo of graphs in the real world



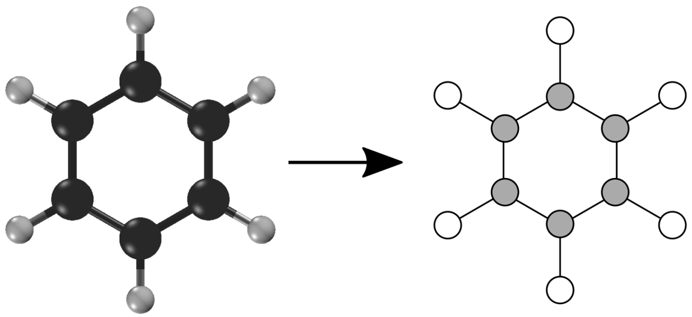
Molecules as graphs
Image from: https://www.oreilly.com/library/view/deep-learning-for/9781492039822/ch04.html

Social networks
Image from: https://medium.com/analytics-vidhya/social-network-analytics-f082f4e21b16

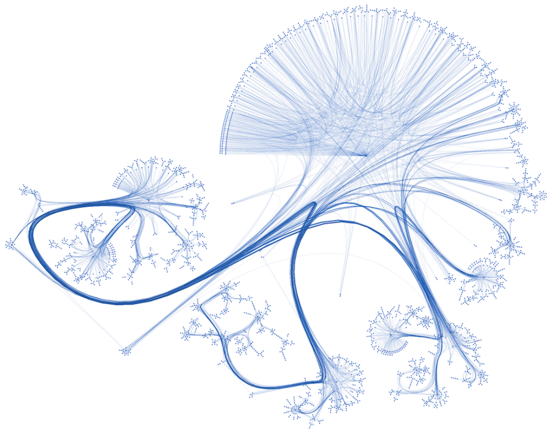


Citation networks
Image from: https://graphsandnetworks.com/the-cora-dataset/

A zoo of graphs in the real world

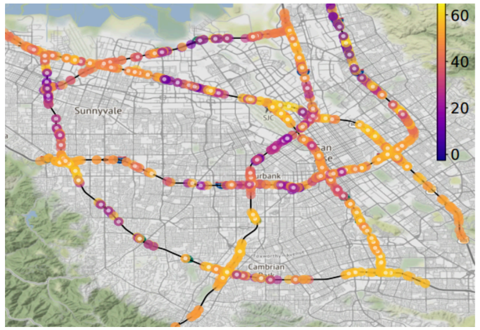


Molecules as graphs
Image from: https://www.oreilly.com/library/view/deep-learning-for/9781492039822/ch04.html



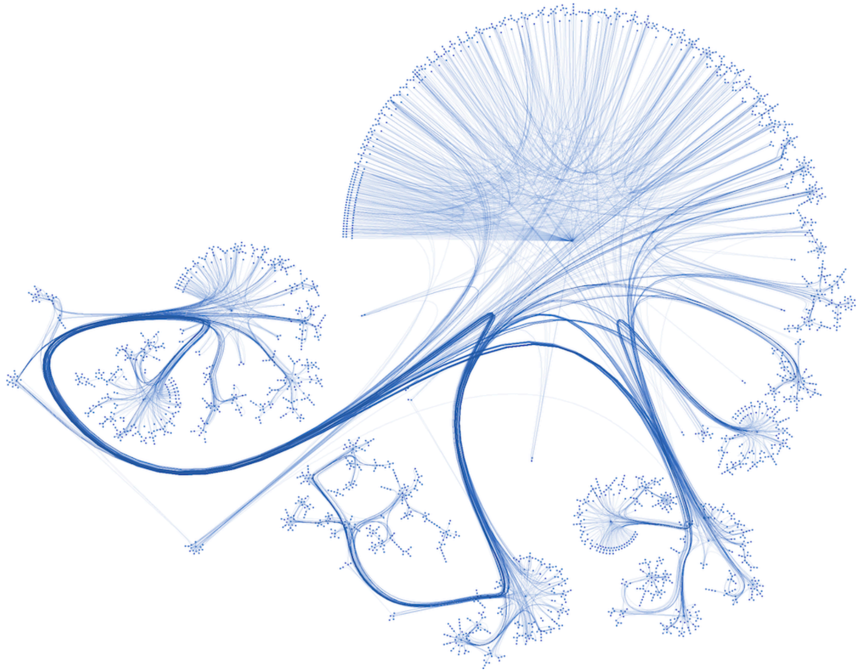
Citation networks
Image from: https://graphsandnetworks.com/the-cora-dataset/

Social networks
Image from: https://medium.com/analytics-vidhya/social-network-analytics-f082f4e21b16



Traffic networks
Image from: http://proceedings.mlr.press/v130/borovitskiy21a/borovitskiy21a.pdf

Example: Cora dataset

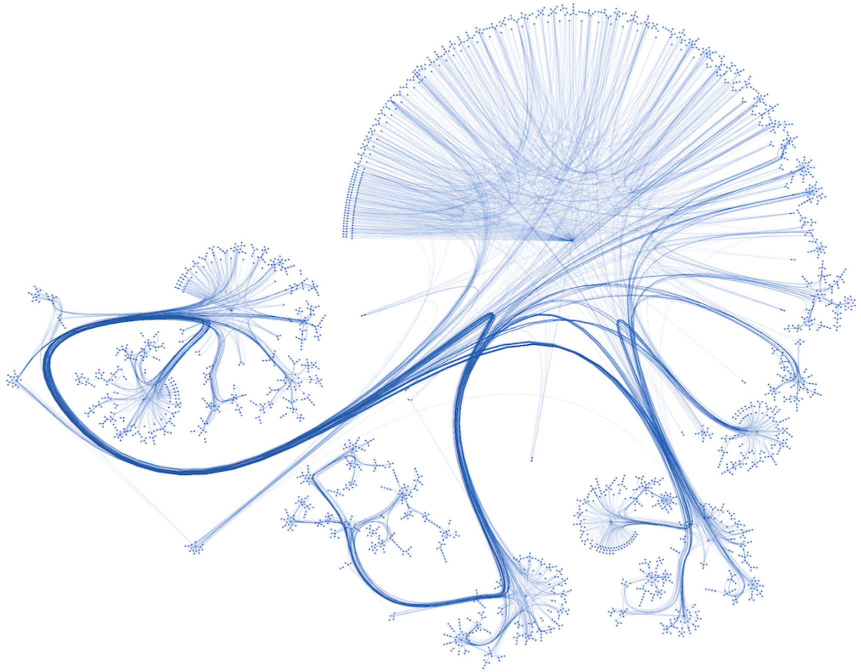


Overview of dataset:

- 2708 ML publications
- 5429 citation links
- Node feature size: 1433
- Seven classes

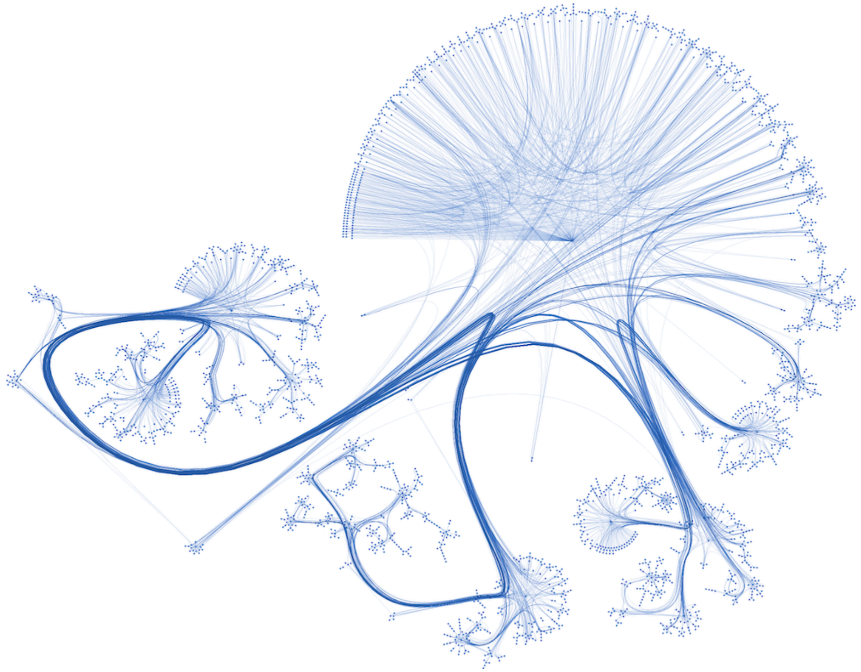
Task: classify nodes according to topic

Example: Cora dataset



Using MLP:

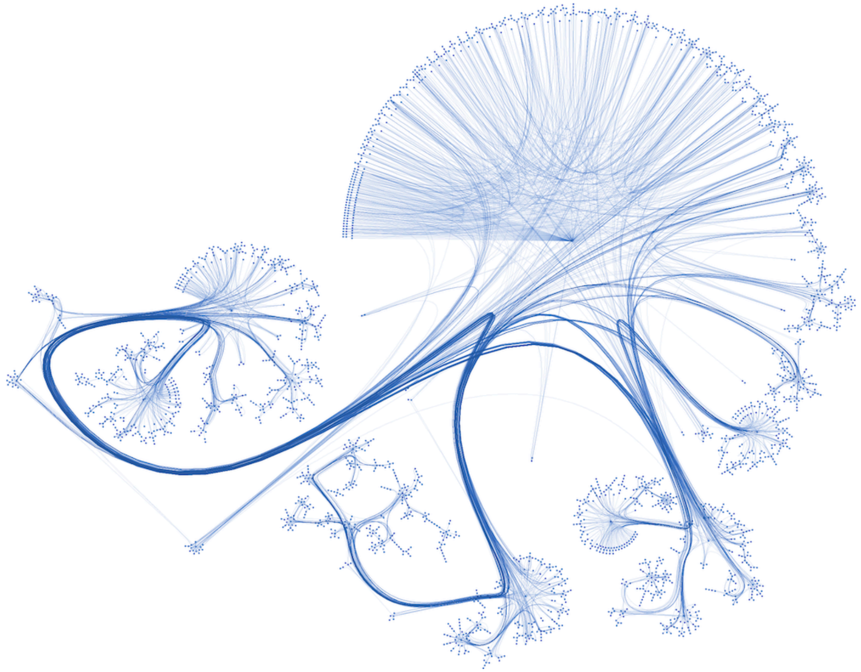
Example: Cora dataset



Using MLP:

Do MLP classification with

Example: Cora dataset

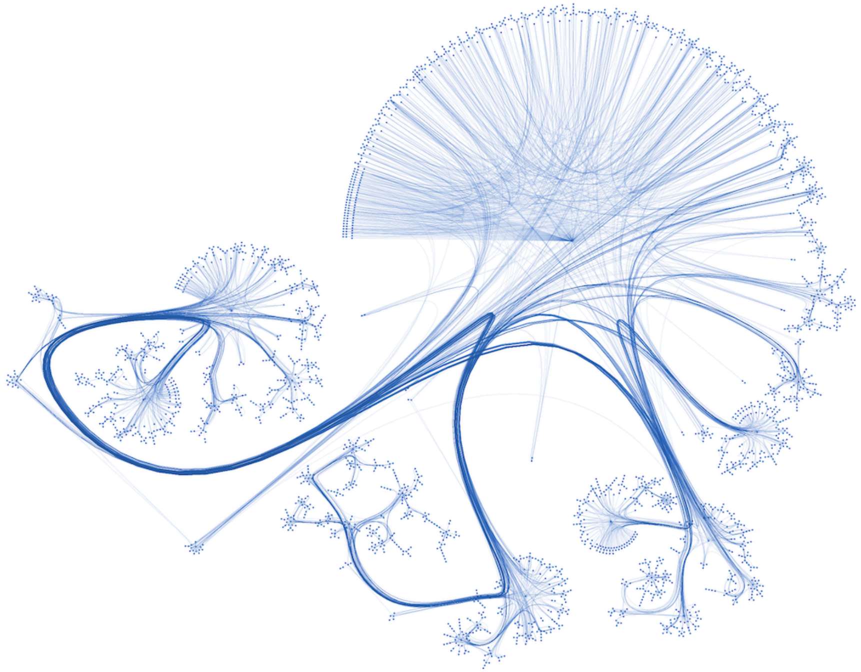


Using MLP:

Do MLP classification with

- Node features as inputs

Example: Cora dataset

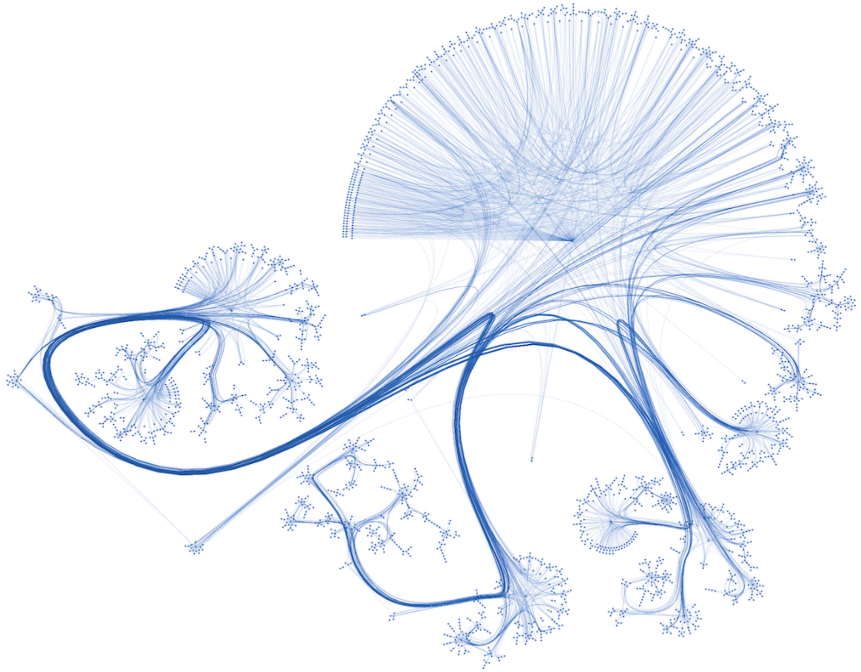


Using MLP:

Do MLP classification with

- Node features as inputs
- Seven topics as outputs

Example: Cora dataset



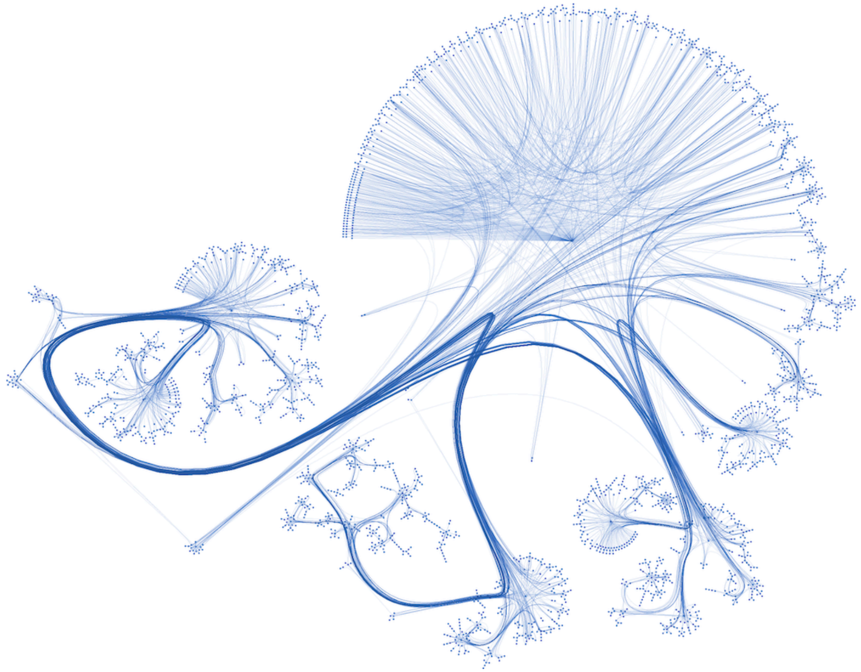
Using MLP:

Do MLP classification with

- Node features as inputs
- Seven topics as outputs

However,

Example: Cora dataset



Using MLP:

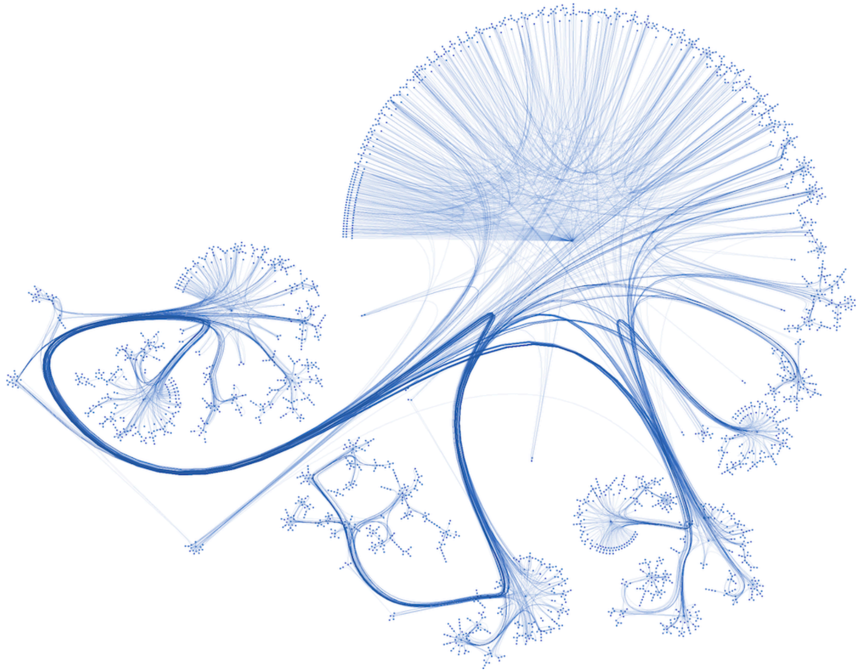
Do MLP classification with

- Node features as inputs
- Seven topics as outputs

However,

- This ignores relational information

Example: Cora dataset



Using MLP:

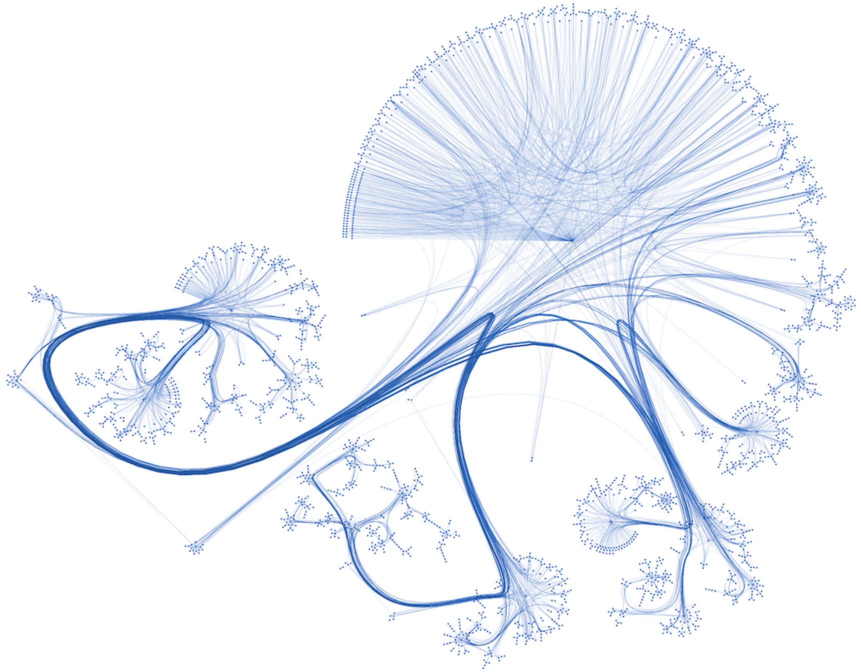
Do MLP classification with

- Node features as inputs
- Seven topics as outputs

However,

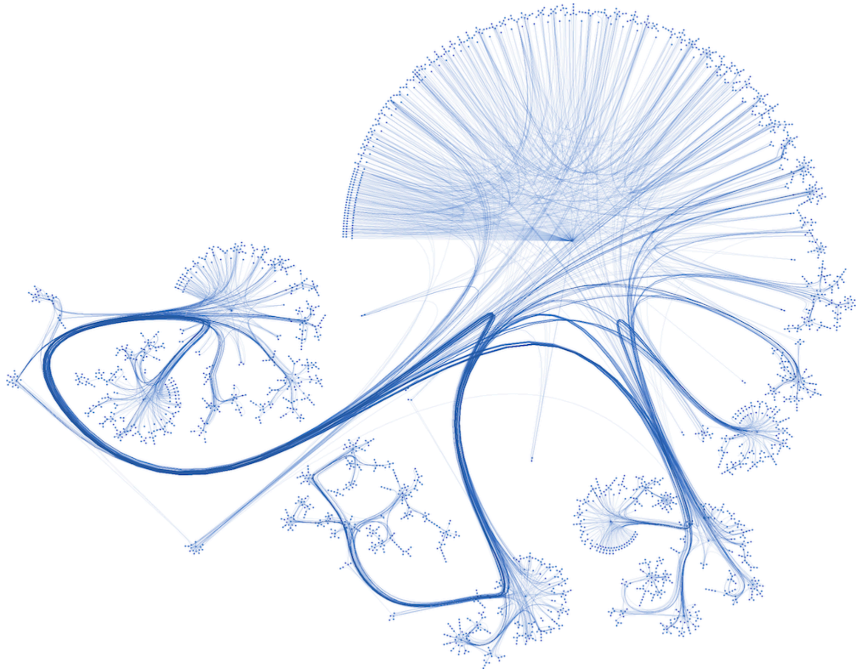
- This ignores relational information
- Data size is small

Example: Cora dataset



Using belief propagation:

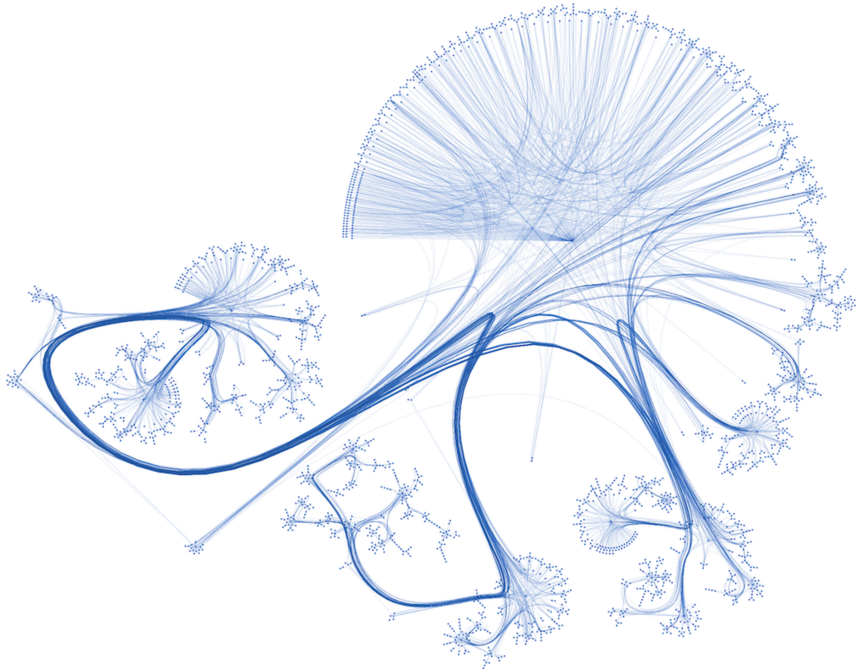
Example: Cora dataset



Using belief propagation:

- Create a MRF with pairwise potential [12]

Example: Cora dataset

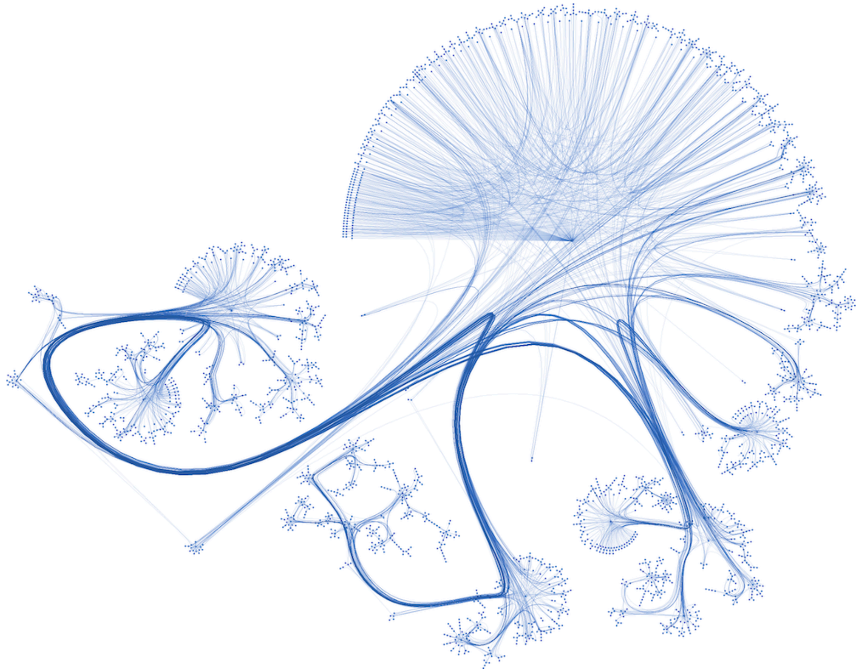


Using belief propagation:

- Create a MRF with pairwise potential [12]

$$\psi_{ij}(x_i, x_j) = \begin{cases} 0.9, & x_i = x_j \\ 0.0166..., & x_i \neq x_j \end{cases}$$

Example: Cora dataset



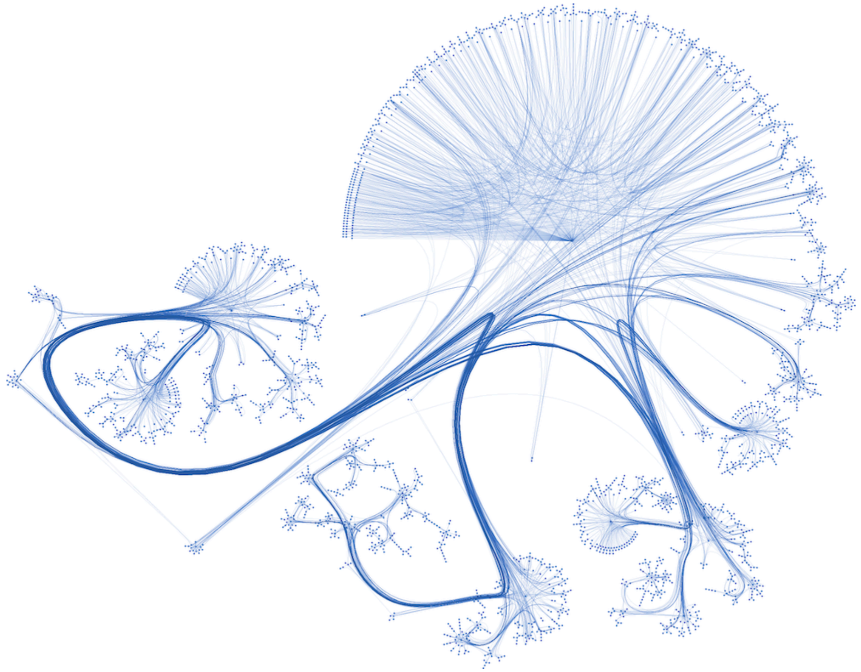
Using belief propagation:

- Create a MRF with pairwise potential [12]

$$\psi_{ij}(x_i, x_j) = \begin{cases} 0.9, & x_i = x_j \\ 0.0166..., & x_i \neq x_j \end{cases}$$

- Perform LBP to compute $p(x_i \,|\, x^{obs})$

Example: Cora dataset



Using belief propagation:

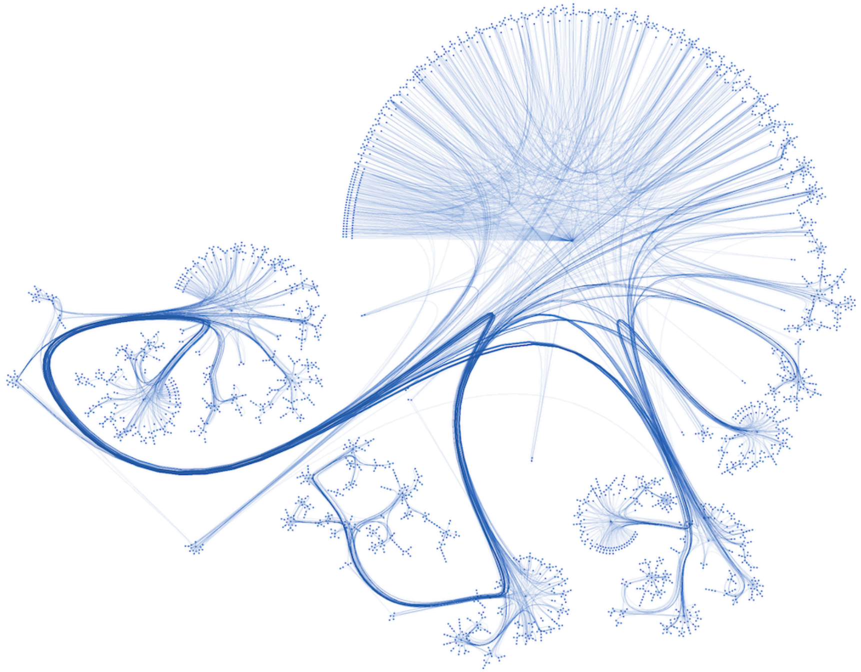
- Create a MRF with pairwise potential [12]

$$\psi_{ij}(x_i, x_j) = \begin{cases} 0.9, & x_i = x_j \\ 0.0166..., & x_i \neq x_j \end{cases}$$

- Perform LBP to compute $p(x_i \mid x^{obs})$

However,

Example: Cora dataset



Using belief propagation:

- Create a MRF with pairwise potential [12]

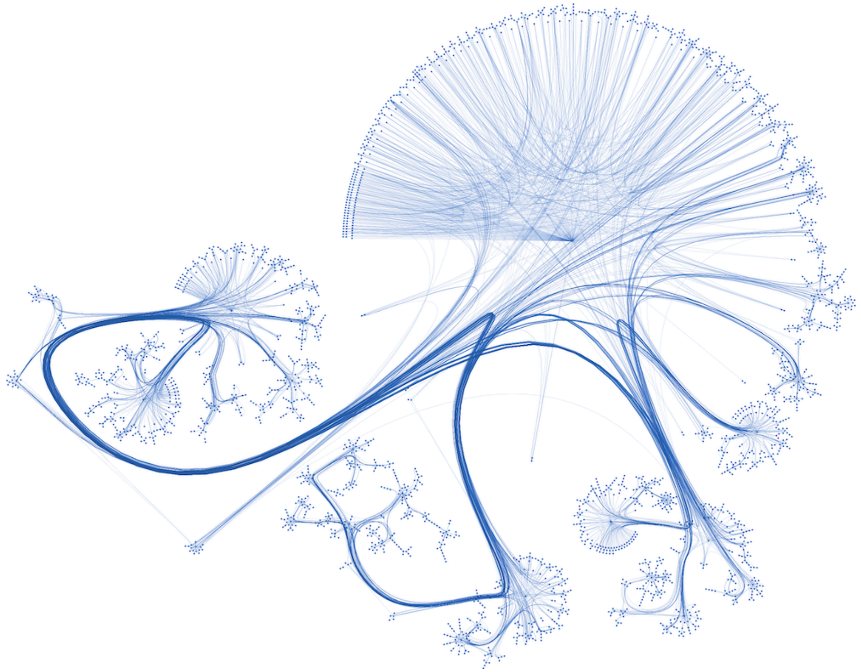
$$\psi_{ij}(x_i, x_j) = \begin{cases} 0.9, & x_i = x_j \\ 0.0166..., & x_i \neq x_j \end{cases}$$

- Perform LBP to compute $p(x_i \mid x^{obs})$

However,

- This does not consider node features

Example: Cora dataset



Using belief propagation:

- Create a MRF with pairwise potential [12]

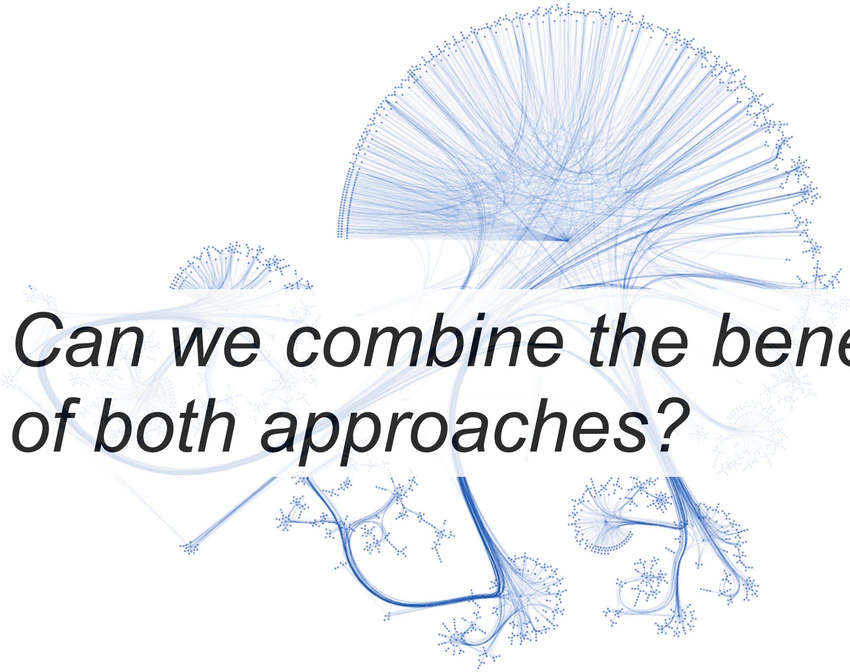
$$\psi_{ij}(x_i, x_j) = \begin{cases} 0.9, & x_i = x_j \\ 0.0166..., & x_i \neq x_j \end{cases}$$

- Perform LBP to compute $p(x_i \mid x^{obs})$

However,

- This does not consider node features
- Pairwise potential is arbitrary

Example: Cora dataset



Can we combine the benefits of both approaches?

Using belief propagation:

- Create a MRF with pairwise potential [12]

$$\psi_{ij}(x_i, x_j) = \begin{cases} 0.9, & x_i = x_j \\ 0.0166..., & x_i \neq x_j \end{cases}$$

- Perform LBP to compute $p(x_i \,|\, x^{obs})$

However,

- This does not consider node features
- Pairwise potential is arbitrary

Convolutional neural networks

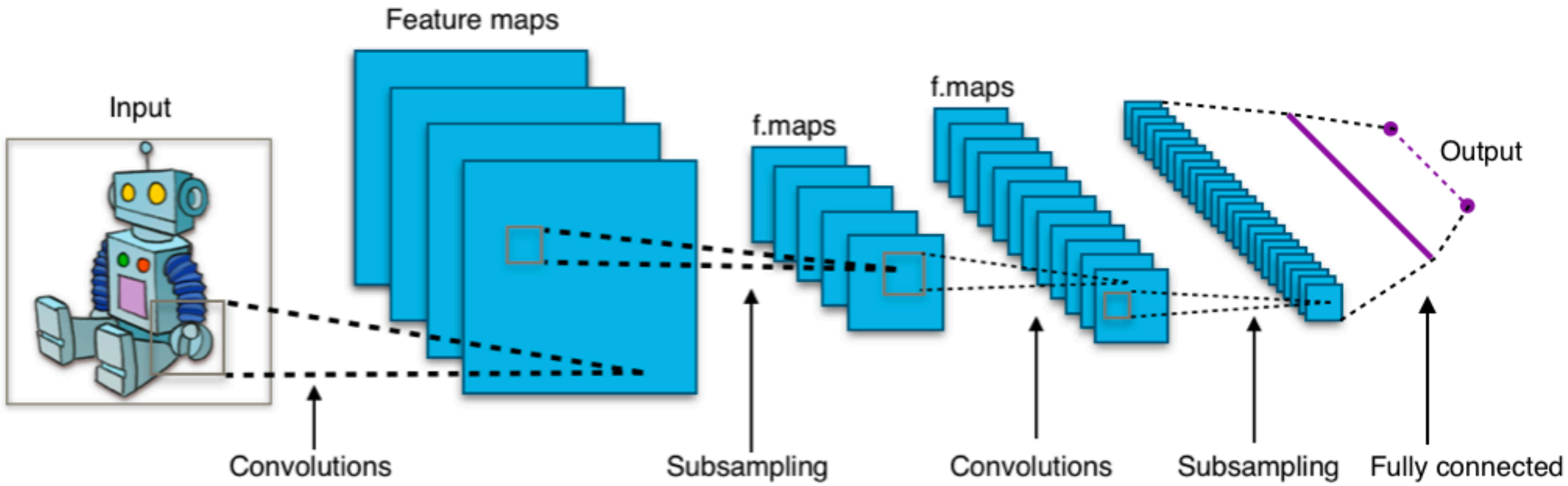


Image from: https://en.wikipedia.org/wiki/Convolutional_neural_network

Convolutional neural networks

- Incorporates inductive bias of grid-inputs

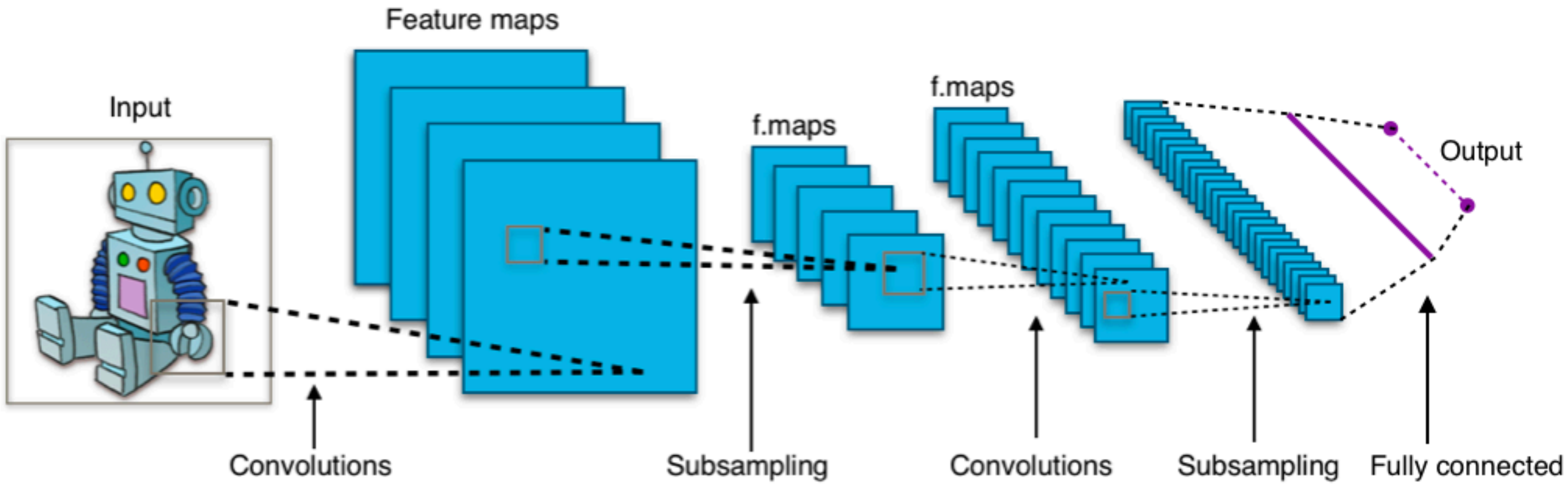


Image from: https://en.wikipedia.org/wiki/Convolutional_neural_network

Convolutional neural networks

- Incorporates inductive bias of grid-inputs
- Sparse connectivity owing to local receptive field

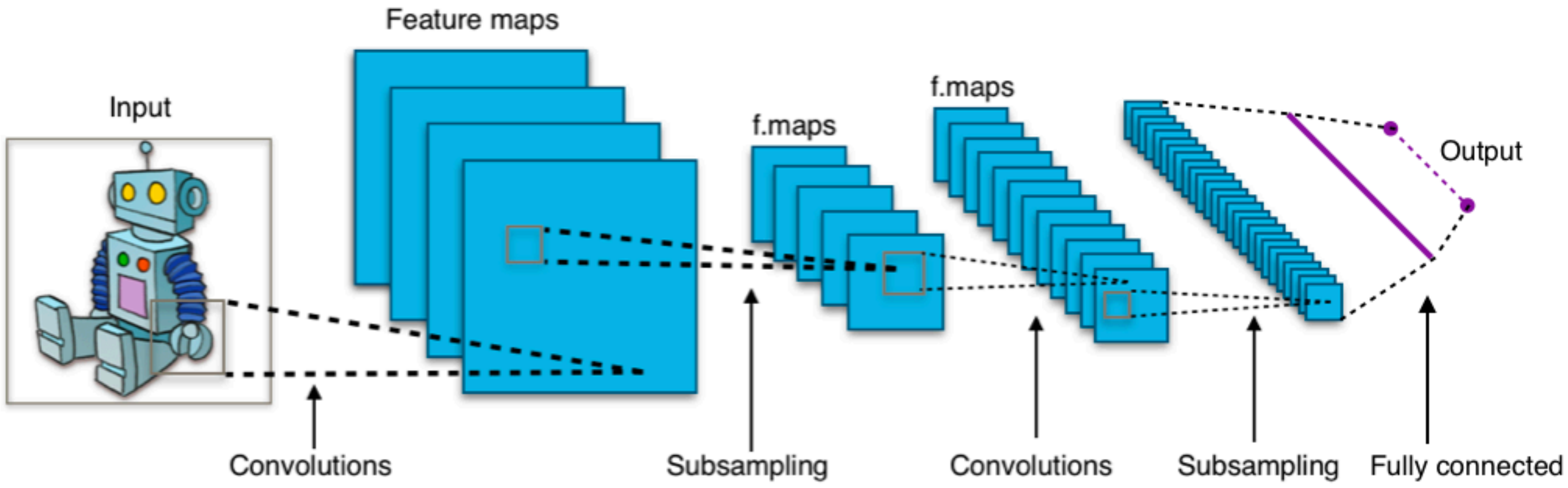


Image from: https://en.wikipedia.org/wiki/Convolutional_neural_network

Convolutional neural networks

- Incorporates inductive bias of grid-inputs
- Sparse connectivity owing to local receptive field
- Shared parameters

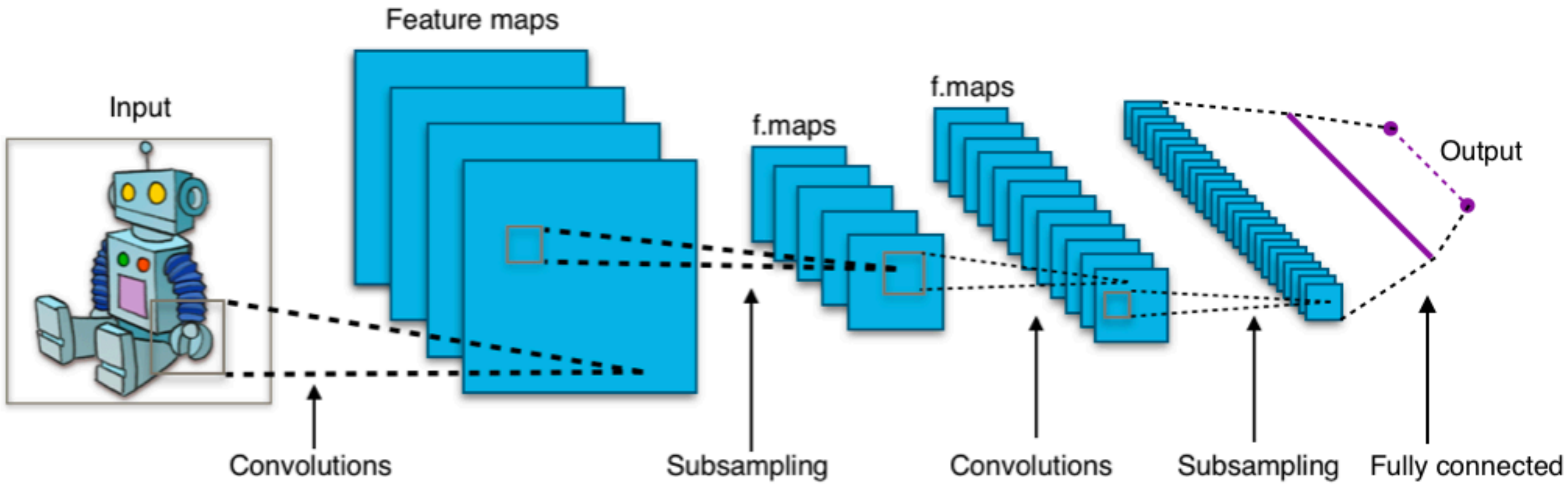


Image from: https://en.wikipedia.org/wiki/Convolutional_neural_network

Criteria for an “ideal” graph NN

Criteria for an "ideal" graph NN

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency

Criteria for an “ideal” graph NN

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency

2. Parameter size independent of input size

Criteria for an "ideal" graph NN

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency

2. Parameter size independent of input size

3. Use local information to construct hidden features

Criteria for an “ideal” graph NN

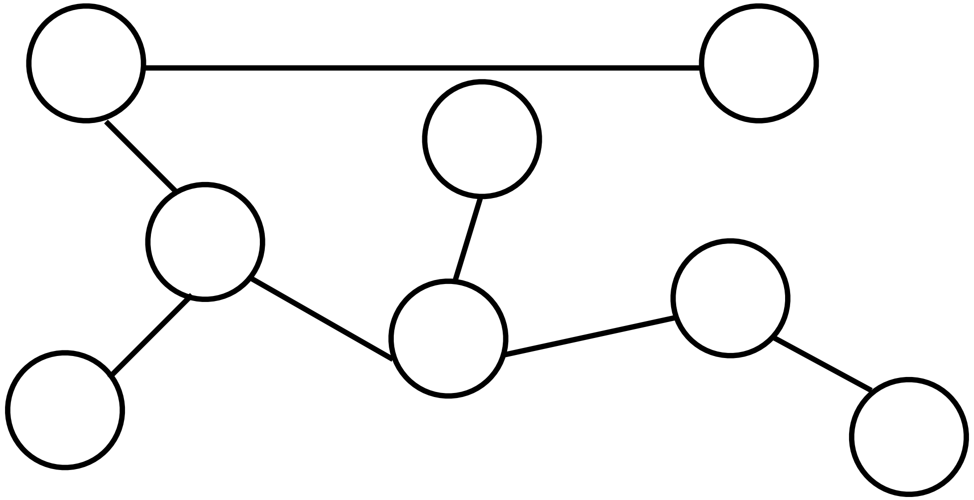
1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency

2. Parameter size independent of input size

3. Use local information to construct hidden features

4. Can use edge features in addition to node features

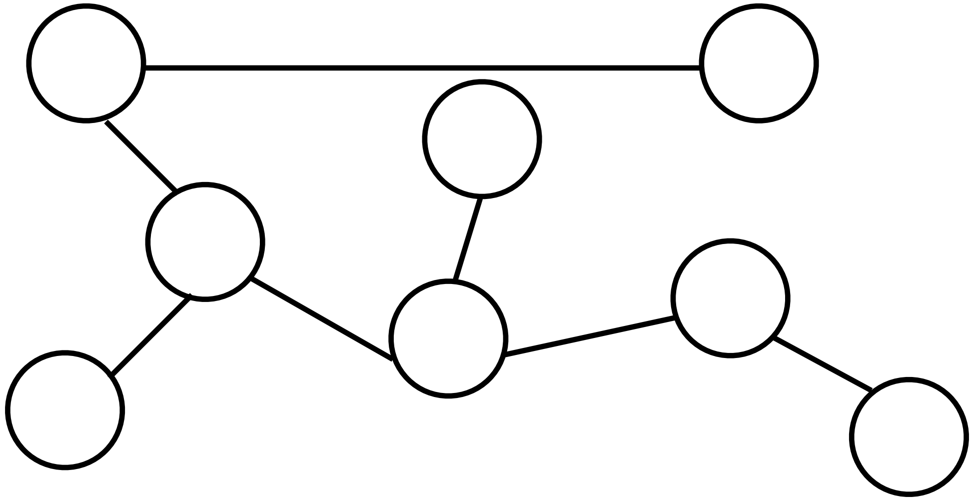
Extending convolutions to graphs?



Can we define convolutions on graphs?

Extending convolutions to graphs?

CNNs are based on discretisation of the *convolution operator*

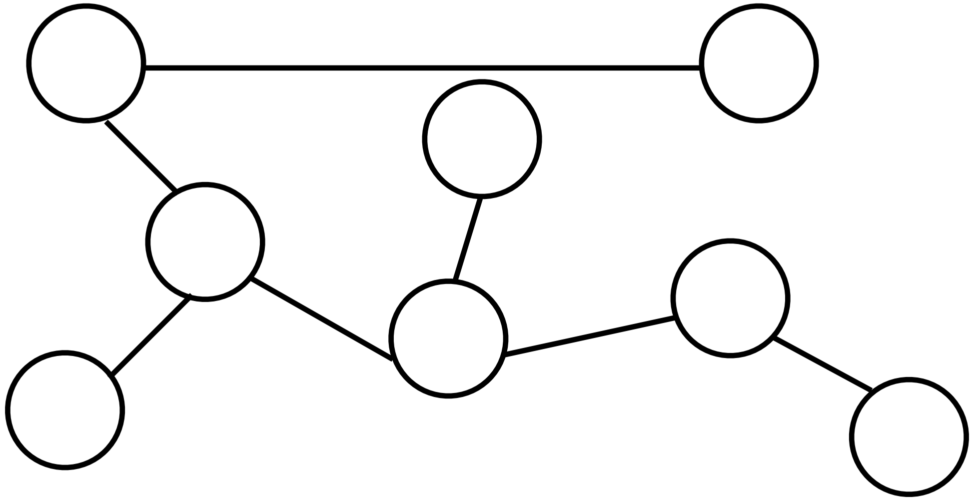


Can we define convolutions on graphs?

Extending convolutions to graphs?

CNNs are based on discretisation of the *convolution operator*

$$f \star \psi_\theta(x) = \int_{\mathbb{R}^2} f(y)\, \psi_\theta(x - y) \mathrm{d}y$$



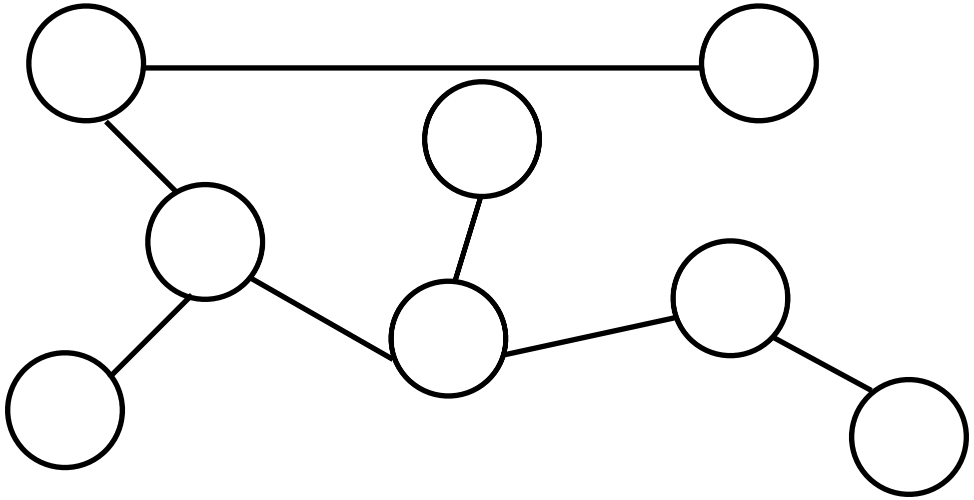
Can we define convolutions on graphs?

Extending convolutions to graphs?

CNNs are based on discretisation of the *convolution operator*

$$f \star \psi_\theta(x) = \int_{\mathbb{R}^2} f(y)\,\psi_\theta(x-y)\mathrm{d}y$$

$$\approx \sum_{y \in \mathbb{Z}^2} f(y)\psi_\theta(x-y)$$

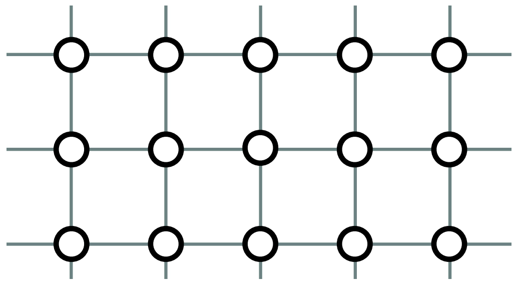


Can we define convolutions on graphs?

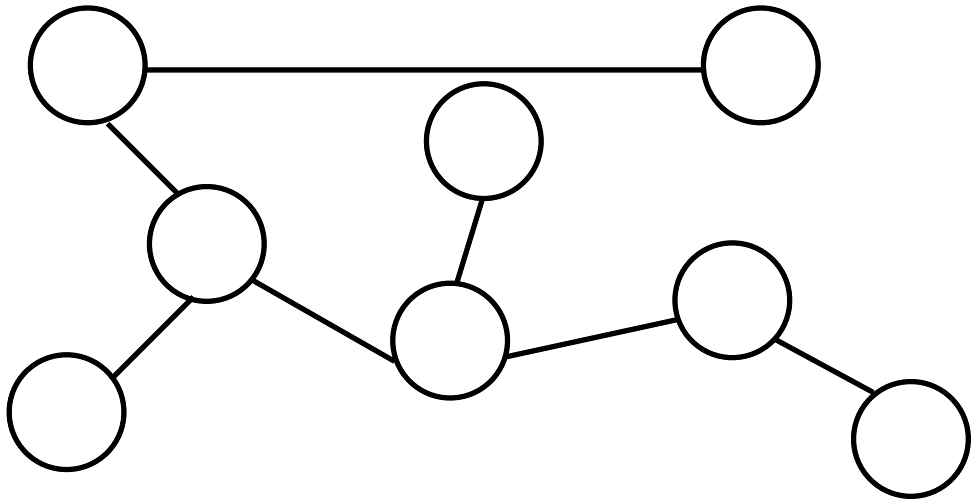
Extending convolutions to graphs?

CNNs are based on discretisation of the *convolution operator*

$$f \star \psi_\theta(x) = \int_{\mathbb{R}^2} f(y)\,\psi_\theta(x-y)\mathrm{d}y$$
$$\approx \sum_{y \in \mathbb{Z}^2} f(y)\psi_\theta(x-y)$$



Convolution applies to grids



Can we define convolutions on graphs?

Spectral graph convolution

Bruna et al. [1] introduced SpectralNet based on the following property of $\star$

$$f \star \psi_\theta(x) = \mathscr{F}^{-1}\left(\mathscr{F}f \odot \mathscr{F}\psi_\theta\right)(x),$$

where $\mathscr{F}$ denotes the Fourier transform.

Spectral graph convolution

Bruna et al. [1] introduced SpectralNet based on the following property of $\star$

$$f \star \psi_\theta(x) = \mathscr{F}^{-1}\left(\mathscr{F}f \odot \mathscr{F}\psi_\theta\right)(x),$$

where $\mathscr{F}$ denotes the Fourier transform.

Observation: Fourier transform can be defined on general graphs!

Spectral graph convolution

Bruna et al. [1] introduced SpectralNet based on the following property of $\star$

$$f \star \psi_\theta(x) = \mathscr{F}^{-1}\left(\mathscr{F}f \odot \mathscr{F}\psi_\theta\right)(x),$$

where $\mathscr{F}$ denotes the Fourier transform.

Observation: Fourier transform can be defined on general graphs!

1. Construct the *graph Laplacian* $\mathbf{L} = \mathbf{D} - \mathbf{A}$
2. Diagonalise $\mathbf{L}$ to get $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$
3. Define $\mathscr{F}\mathbf{f} := \mathbf{U}^\top\mathbf{f}$ and $\mathscr{F}^{-1}\hat{\mathbf{f}} := \mathbf{U}\hat{\mathbf{f}}$

Spectral graph convolution

Bruna et al. [1] introduced SpectralNet based on the following property of $\star$

$$f \star \psi_\theta(x) = \mathscr{F}^{-1}\left(\mathscr{F}f \odot \mathscr{F}\psi_\theta\right)(x),$$

where $\mathscr{F}$ denotes the Fourier transform.

Observation: Fourier transform can be defined on general graphs!

1. Construct the *graph Laplacian* $\mathbf{L} = \mathbf{D} - \mathbf{A}$
2. Diagonalise $\mathbf{L}$ to get $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$
3. Define $\mathscr{F}\mathbf{f} := \mathbf{U}^\top\mathbf{f}$ and $\mathscr{F}^{-1}\hat{\mathbf{f}} := \mathbf{U}\hat{\mathbf{f}}$

Spectral graph convolution

Bruna et al. [1] introduced SpectralNet based on the following property of $\star$

$$f \star \psi_\theta(x) = \mathscr{F}^{-1}\left(\mathscr{F}f \odot \mathscr{F}\psi_\theta\right)(x),$$

where $\mathscr{F}$ denotes the Fourier transform.

Observation: Fourier transform can be defined on general graphs!

1. Construct the *graph Laplacian* $\mathbf{L} = \mathbf{D} - \mathbf{A}$
2. Diagonalise $\mathbf{L}$ to get $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$
3. Define $\mathscr{F}\mathbf{f} := \mathbf{U}^\top\mathbf{f}$ and $\mathscr{F}^{-1}\hat{\mathbf{f}} := \mathbf{U}\hat{\mathbf{f}}$

"Spectral graph convolution"

Spectral graph convolution

Bruna et al. [1] introduced SpectralNet based on the following property of $\star$

$$f \star \psi_\theta(x) = \mathscr{F}^{-1}\left(\mathscr{F}f \odot \underbrace{\mathscr{F}\psi_\theta}_{\hat{\psi}_\theta}\right)(x),$$

where $\mathscr{F}$ denotes the Fourier transform.

Observation: Fourier transform can be defined on general graphs!

1. Construct the *graph Laplacian* $\mathbf{L} = \mathbf{D} - \mathbf{A}$

2. Diagonalise $\mathbf{L}$ to get $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$

3. Define $\mathscr{F}\mathbf{f} := \mathbf{U}^\top\mathbf{f}$ and $\mathscr{F}^{-1}\hat{\mathbf{f}} := \mathbf{U}\hat{\mathbf{f}}$

“Spectral graph convolution”

Spectral graph convolution

Bruna et al. [1] introduced SpectralNet based on the following property of $\star$

$$f \star \psi_\theta(x) = \mathscr{F}^{-1}\left(\mathscr{F}f \odot \underbrace{\mathscr{F}\psi_\theta}_{\hat{\psi}_\theta = \boldsymbol{\theta}}\right)(x),$$

where $\mathscr{F}$ denotes the Fourier transform.

Observation: Fourier transform can be defined on general graphs!

1. Construct the *graph Laplacian* $\mathbf{L} = \mathbf{D} - \mathbf{A}$

2. Diagonalise $\mathbf{L}$ to get $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$

3. Define $\mathscr{F}\mathbf{f} := \mathbf{U}^\top\mathbf{f}$ and $\mathscr{F}^{-1}\hat{\mathbf{f}} := \mathbf{U}\hat{\mathbf{f}}$

"Spectral graph convolution"

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency
   - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size
   - Parameter size is $|V|$
3. Use local information to construct hidden features
   - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features
   - Does not use edge features

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency
   - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size
   - Parameter size is $|V|$
3. Use local information to construct hidden features
   - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features
   - Does not use edge features

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✗
   - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size
   - Parameter size is $|V|$
3. Use local information to construct hidden features
   - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features
   - Does not use edge features

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✗
   - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size
   - Parameter size is $|V|$
3. Use local information to construct hidden features
   - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features
   - Does not use edge features

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✗
   - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size ✗
   - Parameter size is $|V|$
3. Use local information to construct hidden features
   - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features
   - Does not use edge features

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✗
   - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size ✗
   - Parameter size is $|V|$
3. Use local information to construct hidden features
   - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features
   - Does not use edge features

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✗
    - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size ✗
    - Parameter size is $|V|$
3. Use local information to construct hidden features ✗
    - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features
    - Does not use edge features

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✗
   - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size ✗
   - Parameter size is $|V|$
3. Use local information to construct hidden features ✗
   - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features
   - Does not use edge features

Spectral graph convolution

How good is SpectralNet?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✗
   - Computational and storage cost for Fourier transform is $\mathcal{O}(|V|^2)$
2. Parameter size independent of input size ✗
   - Parameter size is $|V|$
3. Use local information to construct hidden features ✗
   - Diagonal features in Fourier space are non-local
4. Can use edge features in addition to node features ✗
   - Does not use edge features

Graph Convolutional Networks

Alternatively, consider a “spatial” approach (Duvenaud et al. [3]):

$$h_{v_i}^{l+1} = \sigma\Big(\sum_{j \in \mathcal{N}_i} h_{v_j}^{l} W_{|\mathcal{N}_i|}^{l} \Big), \quad v_i \in V.$$

Graph Convolutional Networks

Alternatively, consider a "spatial" approach (Duvenaud et al. [3]):

$$h_{v_i}^{l+1} = \sigma\Big(\sum_{j \in \mathcal{N}_i} h_{v_j}^{l} W_{|\mathcal{N}_i|}^{l} \Big), \quad v_i \in V.$$

Kipf and Welling [4] introduced the **Graph Convolutional Network (GCN)**:

$$h_{v_i}^{l+1} = \mathrm{ReLU}\left(\sum_{j \in \mathcal{N}_i} h_{v_j}^{l} \frac{W^l}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}} \right), \quad v_i \in V.$$

Graph Convolutional Networks

Alternatively, consider a "spatial" approach (Duvenaud et al. [3]):

$$h_{v_i}^{l+1} = \sigma\Big(\sum_{j \in \mathcal{N}_i} h_{v_j}^{l} W_{|\mathcal{N}_i|}^{l} \Big), \quad v_i \in V.$$

Kipf and Welling [4] introduced the **Graph Convolutional Network (GCN)**:

$$h_{v_i}^{l+1} = \mathrm{ReLU}\left(\sum_{j \in \mathcal{N}_i} h_{v_j}^{l} \frac{W^l}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}} \right), \quad v_i \in V.$$

- Works well in practice

Graph Convolutional Networks

Alternatively, consider a "spatial" approach (Duvenaud et al. [3]):

$$h_{v_i}^{l+1} = \sigma\Big(\sum_{j\in\mathcal{N}_i} h_{v_j}^{l}\, W_{|\mathcal{N}_i|}^{l}\Big), \quad v_i \in V.$$

Kipf and Welling [4] introduced the **Graph Convolutional Network (GCN)**:

$$h_{v_i}^{l+1} = \mathrm{ReLU}\left(\sum_{j\in\mathcal{N}_i} h_{v_j}^{l} \frac{W^l}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}\right), \quad v_i \in V.$$

- Works well in practice
- Can be derived from *ChebNet* [2], a variant of spectral graph convolution

Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency
   - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h^l_{v_j} W^l$ performed $|V|$ times)
   - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size
   - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C \times F}$
3. Use local information to construct hidden features
   - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features
   - Does not use edge features in original formulation

Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency
   - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h^l_{v_j} W^l$ performed $|V|$ times)
   - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size
   - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C \times F}$
3. Use local information to construct hidden features
   - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features
   - Does not use edge features in original formulation

Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✓
   - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h^l_{v_j} W^l$ performed $|V|$ times)
   - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size
   - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C \times F}$
3. Use local information to construct hidden features
   - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features
   - Does not use edge features in original formulation

Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✓
   - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h^l_{v_j} W^l$ performed $|V|$ times)
   - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size
   - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C \times F}$
3. Use local information to construct hidden features
   - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features
   - Does not use edge features in original formulation

Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✓
   - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h_{v_j}^l W^l$ performed $|V|$ times)
   - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size ✓
   - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C \times F}$
3. Use local information to construct hidden features
   - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features
   - Does not use edge features in original formulation

Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✓
   - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h^l_{v_j} W^l$ performed $|V|$ times)
   - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size ✓
   - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C \times F}$
3. Use local information to construct hidden features
   - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features
   - Does not use edge features in original formulation

Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✓
    - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h_{v_j}^l W^l$ performed $|V|$ times)
    - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size ✓
    - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C\times F}$
3. Use local information to construct hidden features ✓
    - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features
    - Does not use edge features in original formulation

Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✓
   - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h^l_{v_j} W^l$ performed $|V|$ times)
   - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size ✓
   - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C \times F}$
3. Use local information to construct hidden features ✓
   - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features
   - Does not use edge features in original formulation

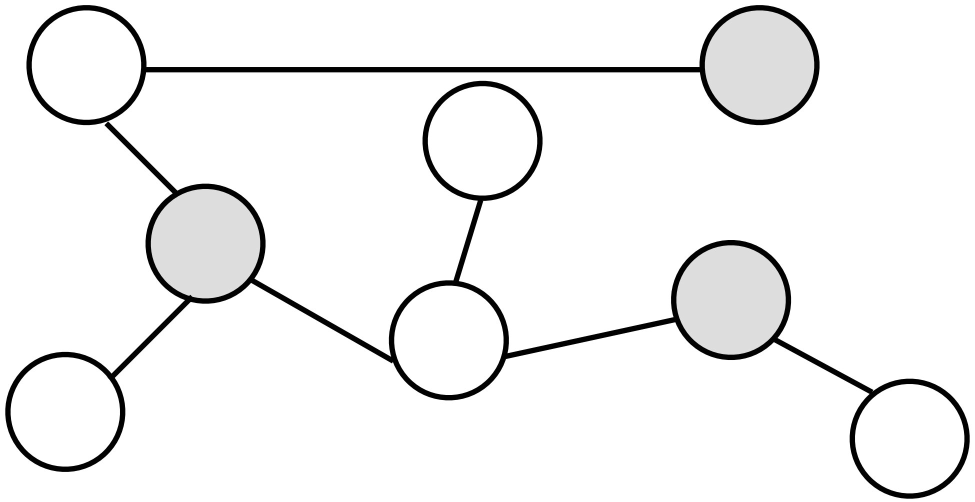
Graph Convolutional Networks

How good is GCN?

1. $\mathcal{O}(|V| + |E|)$ computational and storage efficiency ✓
   - Computational cost is $\mathcal{O}(|V|CF)$ (multiplication $h^l_{v_j} W^l$ performed $|V|$ times)
   - Storage cost is $\mathcal{O}(|E|)$ (to store adjacency matrix $\mathbf{A}$)
2. Parameter size independent of input size ✓
   - Parameter size is $\mathcal{O}(CF)$ per layer to store $W^l \in \mathbb{R}^{C \times F}$
3. Use local information to construct hidden features ✓
   - By construction, hidden features only depend on local neighbours
4. Can use edge features in addition to node features ✗
   - Does not use edge features in original formulation

Semi-supervised learning

- Applies when the number of labelled datapoints are *small*
- But relations between labelled and unlabelled data exist



Semi-supervised learning

Experiment with Cora dataset:

- Use only 140 nodes for training data
- 1000 nodes for testing

Train with cross-entropy loss over labelled data $\mathscr{D}_L$ (i.e. training data):

$$L = -\sum_{(y,X)\in\mathscr{D}_L} y \log \mathrm{GCN}(X).$$

Semi-supervised learning

Experiment with Cora dataset:

- Use only 140 nodes for training data
- 1000 nodes for testing

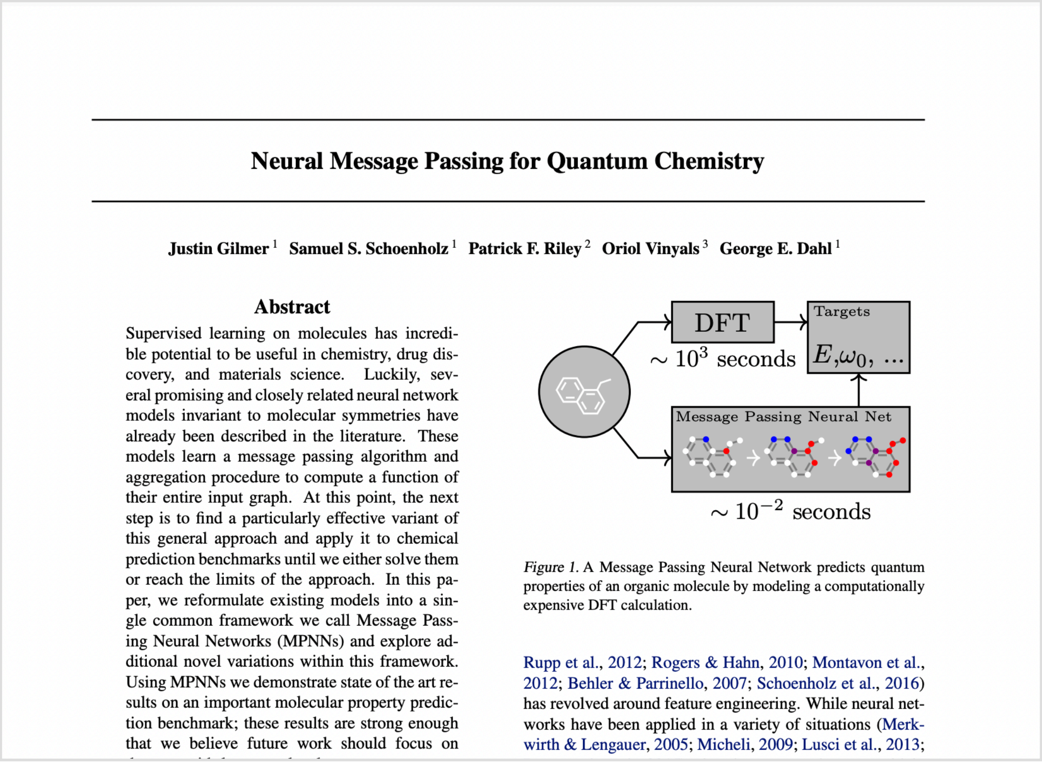
Train with cross-entropy loss over labelled data $\mathscr{D}_L$ (i.e. training data):

$$L = -\sum_{(y,X)\in\mathscr{D}_L} y \log \mathrm{GCN}(X).$$

Kipf and Welling [4] reports accuracy of:

- $81.5\,\%$ using GCN
- $55.1\,\%$ using MLP

Message Passing Neural Networks



Neural Message Passing for Quantum Chemistry

Justin Gilmer [1] Samuel S. Schoenholz [1] Patrick F. Riley [2] Oriol Vinyals [3] George E. Dahl [1]

Abstract

Supervised learning on molecules has incredible potential to be useful in chemistry, drug discovery, and materials science. Luckily, several promising and closely related neural network models invariant to molecular symmetries have already been described in the literature. These models learn a message passing algorithm and aggregation procedure to compute a function of their entire input graph. At this point, the next step is to find a particularly effective variant of this general approach and apply it to chemical prediction benchmarks until we either solve them or reach the limits of the approach. In this paper, we reformulate existing models into a single common framework we call Message Passing Neural Networks (MPNNs) and explore additional novel variations within this framework. Using MPNNs we demonstrate state of the art results on an important molecular property prediction benchmark; these results are strong enough that we believe future work should focus on

Figure 1. A Message Passing Neural Network predicts quantum properties of an organic molecule by modeling a computationally expensive DFT calculation.

Rupp et al., 2012; Rogers & Hahn, 2010; Montavon et al., 2012; Behler & Parrinello, 2007; Schoenholz et al., 2016) has revolved around feature engineering. While neural networks have been applied in a variety of situations (Merkwirth & Lengauer, 2005; Micheli, 2009; Lusci et al., 2013;

Gilmer et. al. [5] (ICML, 2017)

- Developed to predict properties of molecules
- Introduces a general framework for learning features on graphs based on message passing
- Can handle graph data containing both node and edge features

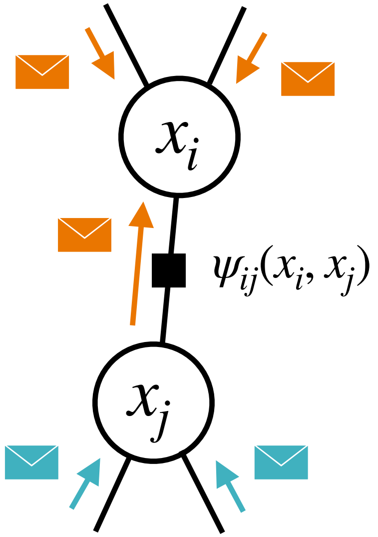
Recall the message passing protocol in BP:

Message update:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\ldots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

State update:

$$p(x_i) = \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$



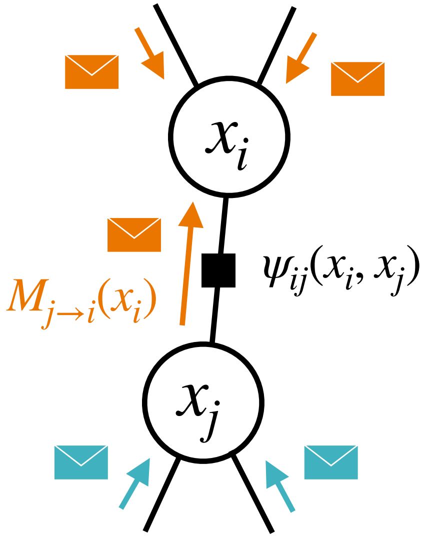
Recall the message passing protocol in BP:

Message update:

$$M_{j\to i}(x_i) = \sum_{x_j \in \{1,\dots,K\}} \psi_{ij}(x_i, x_j)\, \psi_j(x_j) \prod_{k\sim j, k\neq i} M_{k\to j}(x_j),$$

State update:

$$p(x_i) = \psi_i(x_i) \prod_{j\sim i} M_{j\to i}(x_i).$$



Message passing in MPNN [6]:

Message update:

$$M_{j\to i}^{l} = M_{\theta}^{l}(h_{v_i}^{l}, h_{v_j}^{l}, e_{ij}),$$

State update:

$$h_{v_i}^{l+1} = U_{\theta}^{l}(h_{v_i}^{l}, \square_{j\sim i}\, M_{j\to i}^{l}).$$

Readout:

$$y = R_{\theta}(\{h_{v_i}^{L} \,|\, v_i \in V\}).$$

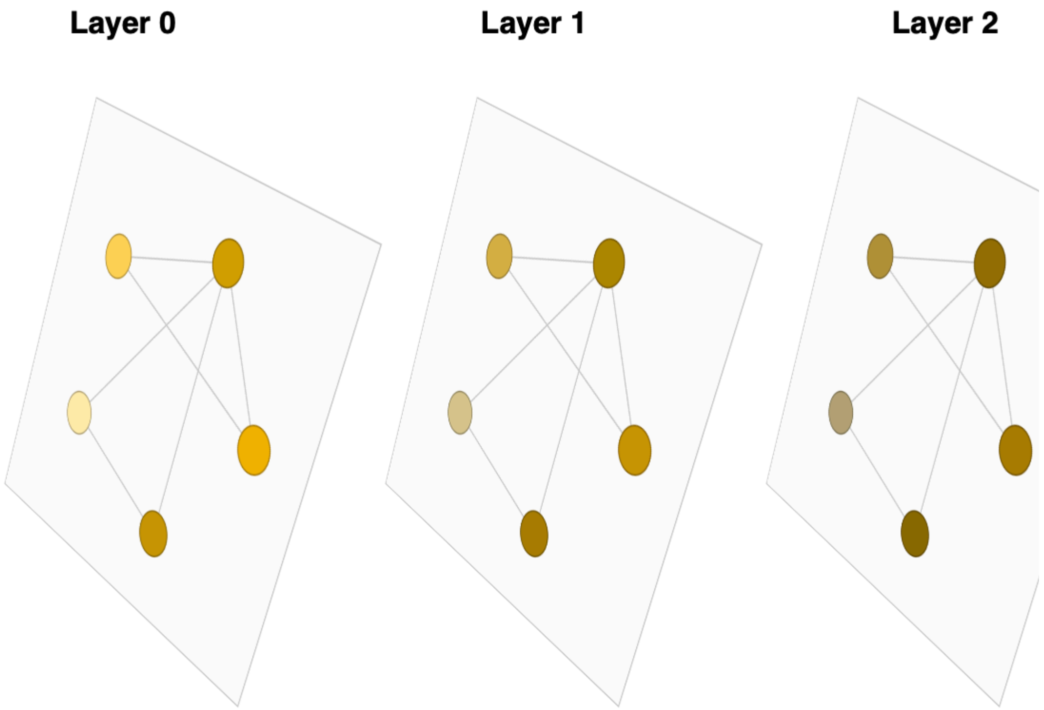


Image from: https://distill.pub/2021/gnn-intro/

Message passing in MPNN [6]:

Message update:

$$M^l_{j\to i} = M^l_\theta(h^l_{v_i}, h^l_{v_j}, e_{ij}),$$

State update:

$$h^{l+1}_{v_i} = U^l_\theta(h^l_{v_i}, \square_{j\sim i}\, M^l_{j\to i}).$$

Readout:

$$y = R_\theta(\{h^L_{v_i} \,|\, v_i \in V\}).$$

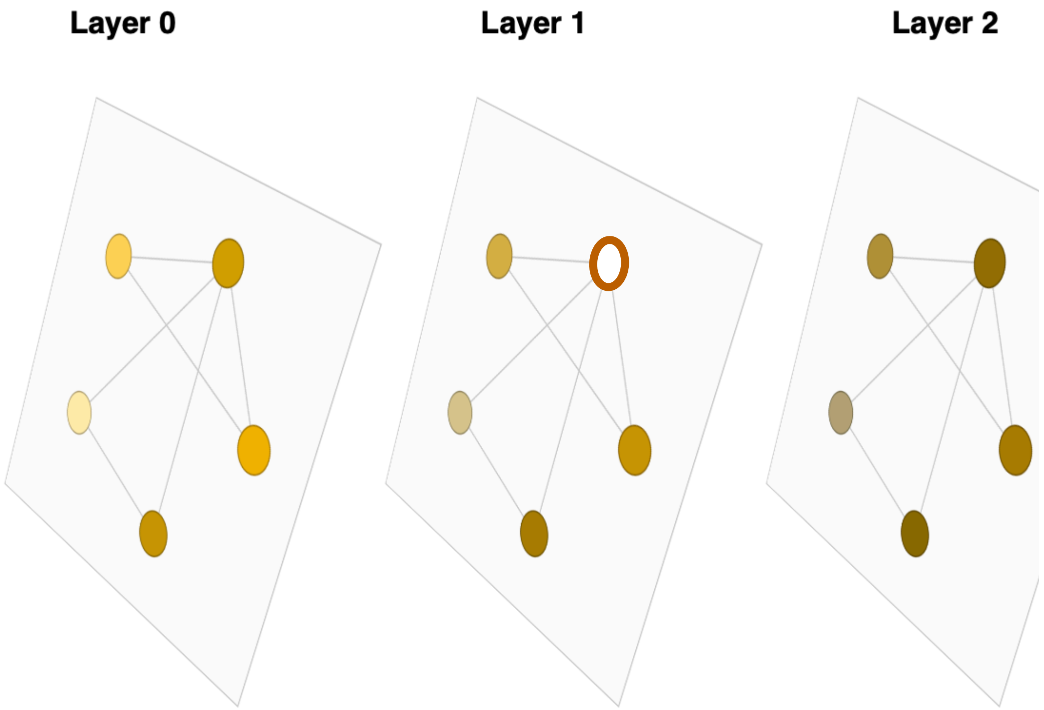


Image from: https://distill.pub/2021/gnn-intro/

Message passing in MPNN [6]:

Message update:

$$M^l_{j\to i} = M^l_\theta(h^l_{v_i}, h^l_{v_j}, e_{ij}),$$

State update:

$$h^{l+1}_{v_i} = U^l_\theta(h^l_{v_i}, \square_{j\sim i}\, M^l_{j\to i}).$$

Readout:

$$y = R_\theta(\{h^L_{v_i} \,|\, v_i \in V\}).$$

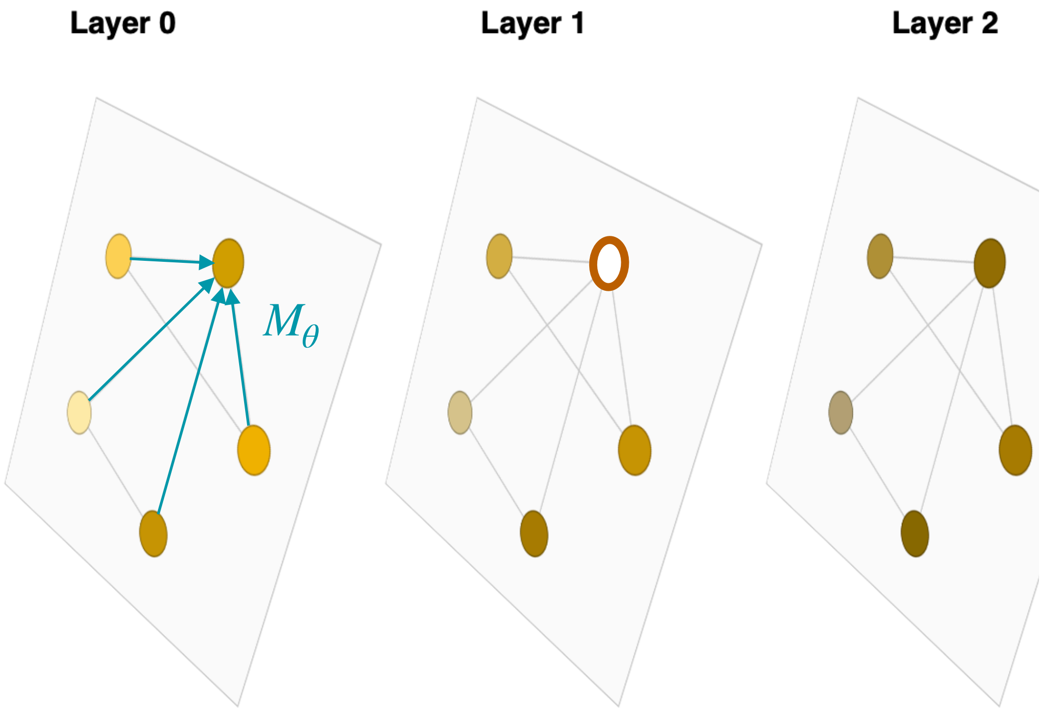


Image from: https://distill.pub/2021/gnn-intro/

Message passing in MPNN [6]:

Message update:

$$M^l_{j\to i} = M^l_\theta(h^l_{v_i}, h^l_{v_j}, e_{ij}),$$

State update:

$$h^{l+1}_{v_i} = U^l_\theta(h^l_{v_i}, \square_{j\sim i}\, M^l_{j\to i}).$$

Readout:

$$y = R_\theta(\{h^L_{v_i} \,|\, v_i \in V\}).$$

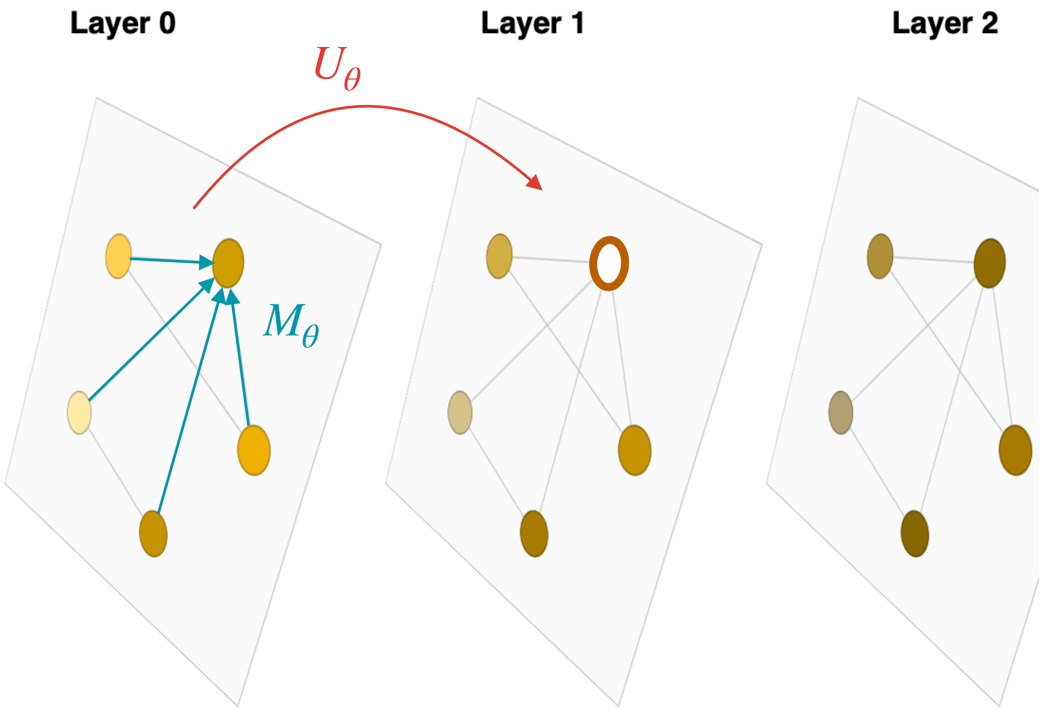


Image from: https://distill.pub/2021/gnn-intro/

Message passing in MPNN [6]:

Message update:

$$M^l_{j\to i} = M^l_\theta(h^l_{v_i}, h^l_{v_j}, e_{ij}),$$

State update:

$$h^{l+1}_{v_i} = U^l_\theta(h^l_{v_i}, \square_{j\sim i} M^l_{j\to i}).$$

Readout:

$$y = R_\theta(\{h^L_{v_i} \,|\, v_i \in V\}).$$

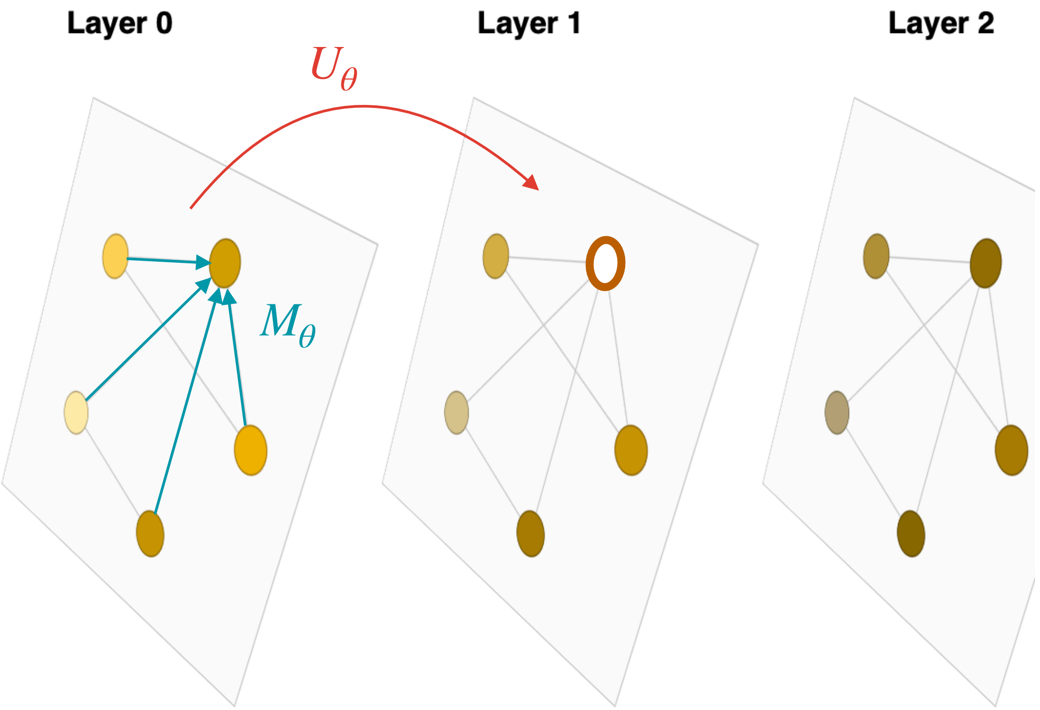


Image from: https://distill.pub/2021/gnn-intro/

Most GNN architectures can be expressed as an MPNN!

Example 1: GCNs as MPNN

Recall the GCN architecture:

$$h_{v_i}^{l+1} = \mathrm{ReLU}\left(\sum_{j \in \mathcal{N}_i} h_{v_j}^{l} \frac{W^l}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}} \right), \quad v_i \in V.$$

Example 1: GCNs as MPNN

Recall the GCN architecture:

$$h_{v_i}^{l+1} = \mathrm{ReLU}\left(\sum_{j \in \mathcal{N}_i} h_{v_j}^{l} \frac{W^l}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}} \right), \quad v_i \in V.$$

This can be expressed as an MPNN with:

- $M_\theta^l(h_{v_i}^l, h_{v_j}^l, e_{ij}) = \dfrac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}} h_{v_j}^l$
- $U_\theta^l(h_{v_i}^l, \square_{j\sim i} M_{j\to i}^l) = \mathrm{ReLU}\left(\left(\dfrac{1}{|\mathcal{N}_i|} h_{v_i}^l + \sum_{j\sim i} M_\theta^l(h_{v_i}^l, h_{v_j}^l, e_{ij}) \right) W^l \right)$

Example 2: MPNN in Gilmer et al. [5]

The original work of Gilmer et al. [5] used the following MPNN model

- $M_\theta^l(h_{v_i}^l, h_{v_j}^l, e_{ij}) = \mathrm{MLP}(e_{ij})\, h_{v_j}^l$
- $U_\theta^l(h_{v_i}^l, \square_{j\sim i} M_{j\to i}^l) = \mathrm{GRU}\left(h_{v_i}^l, \sum_{j\sim i} M_{j\to i}^l\right)$

to predict 13 quantum properties of molecules in the QM9 dataset.

Example 2: MPNN in Gilmer et al. [5]

The original work of Gilmer et al. [5] used the following MPNN model

- $M_\theta^l(h_{v_i}^l, h_{v_j}^l, e_{ij}) = \mathrm{MLP}(e_{ij})\, h_{v_j}^l$
- $U_\theta^l(h_{v_i}^l, \square_{j\sim i} M_{j\to i}^l) = \mathrm{GRU}\left(h_{v_i}^l, \sum_{j\sim i} M_{j\to i}^l\right)$

to predict 13 quantum properties of molecules in the QM9 dataset.

Model performs extremely well with 11 out of 13 properties reaching "chemical accuracy".

Example 3: Transformers

MPNNs also encompass the transformer [9] model:

- $M^l_\theta(h^l_{v_i}, h^l_{v_j}, e_{ij}) = \mathrm{MultiheadAttention}(h^l_{v_i}, h^l_{v_j})$

$$= \left\{ w^k_{ij}(h^l_{v_i}, h^l_{v_j}), V^k_j(h^l_{v_j}) \right\}_{k=1}^{K}$$

- $U^l_\theta(h^l_{v_i}, \square_{j\sim i} M^l_{j\to i}) = \mathrm{LN}\Big(\mathrm{MLP}\Big(\mathrm{LN}\Big(\sum_{j\sim i} w^k_{ij} V^k_j\Big)\Big)\Big)$

where the graph is assumed to be *fully-connected.*

(See blogpost [8] for more details)

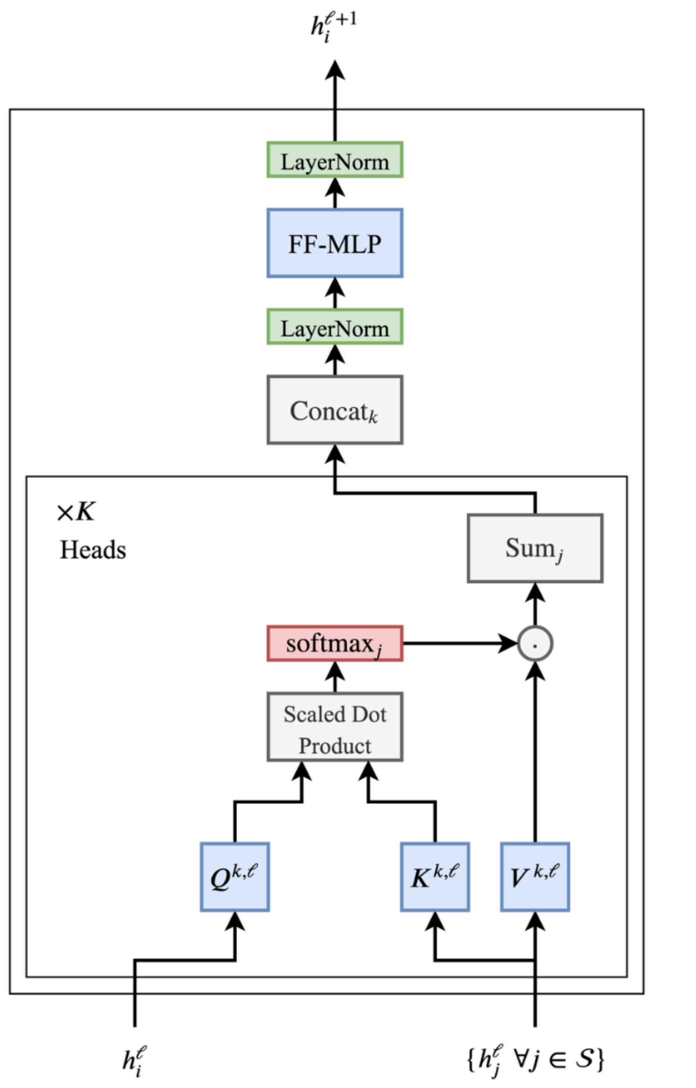


Image from [8]

Comparison of MPNN with LBP

LBP	MPNN
Bayesian. Coupling between neighbours arise from prior knowledge of model. Message passing rule follows from laws of probability.	**Frequentist.** Message and state update rules are learned from data to obtain useful feature representations.
Iterative. States are updated iteratively to obtain better estimates of marginals.	**Deep.** Uses the power of deep learning to extract increasingly complex features with depth.
Interpretable. Prior assumptions are usually quite simple, making predictions interpretable.	**Flexible.** Processes high-dimensional node and edge features easily to model complex relations between inputs and outputs.

Comparison of MPNN with LBP

LBP	MPNN
Bayesian. Coupling between neighbours arise from prior knowledge of model. Message passing rule follows from laws of probability.	**Frequentist.** Message and state update rules are learned from data to obtain useful feature representations.
Iterative. States are updated iteratively to obtain better estimates of marginals.	**Deep.** Uses the power of deep learning to extract increasingly complex features with depth.
Interpretable. Prior assumptions are usually quite simple, making predictions interpretable.	**Flexible.** Processes high-dimensional node and edge features easily to model complex relations between inputs and outputs.

Many recent works aim to combine benefits of both approaches ([10] - [14])!

References

[1] Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. "Spectral networks and deep locally connected networks on graphs." *ICLR*, 2014.

[2] Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering." *NeurIPS, 2016*.

[3] Duvenaud, David K., Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. "Convolutional networks on graphs for learning molecular fingerprints." *NeurIPS,* 2015.

[4] Welling, Max, and Thomas N. Kipf. "Semi-supervised classification with graph convolutional networks." *ICLR,* 2017.

[5] Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. "Neural message passing for quantum chemistry." *ICML*, 2017.

References

[6] PyTorch Geometric. “Creating Message Passing Networks”: https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html

[7] Sanchez-Lengeling, Benjamin, Emily Reif, Adam Pearce and Alexander B. Wiltschko, "A Gentle Introduction to Graph Neural Networks”. *Distill*, 2021.

[8] Joshi, Chaitanya K. "Transformers are Graph Neural Networks", *The Gradient*, 2020.

[9] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." *NeurIPS,* 2017.

[10] Ying, Zhitao, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. "GNNexplainer: Generating explanations for graph neural networks." *NeurIPS,* 2019.

References

[11] Yoon, KiJung, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. "Inference in probabilistic graphical models by graph neural networks." *53rd Asilomar Conference on Signals, Systems, and Computers,* 2019.

[12] Kuck, Jonathan, Shuvam Chakraborty, Hao Tang, Rachel Luo, Jiaming Song, Ashish Sabharwal, and Stefano Ermon. "Belief propagation neural networks." *NeurIPS,* 2020.

[13] Wang, Binghui, Jinyuan Jia, and Neil Zhenqiang Gong. "Semi-Supervised Node Classification on Graphs: Markov Random Fields vs. Graph Neural Networks." *AAAI*, 2021.

[14] Satorras, Victor Garcia, and Max Welling. "Neural enhanced belief propagation on factor graphs." *AISTATS*, 2021.

[15] Hua, Chenqing, Sitao Luan, Qian Zhang and Jie Fu. "Graph Neural Networks Intersect Probabilistic Graphical Models: A Survey." *arXiv preprint arXiv:2206.06089,* 2022.