Welcome to this course module!!!
Graphical Model Inference

Fredrik Lindsten, Linköping University

2023-10-30

Simultaneous Localization and Mapping (SLAM)

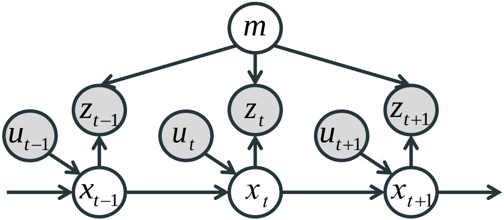
ex) Robot co-operating with humans in indoor environment

The robot's task:

- Figure out where it is (localization)
- Figure out what the environment looks like (mapping)

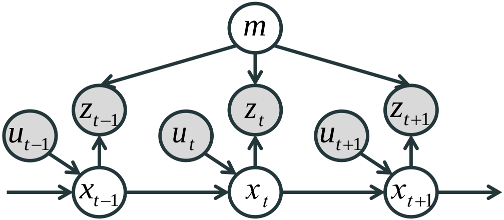
SLAM graphical model

Use a **graphical model** to illustrate dependencies between robot position (x_t), position of landmarks (m) and control/sensory input (u_t, z_t).



SLAM graphical model

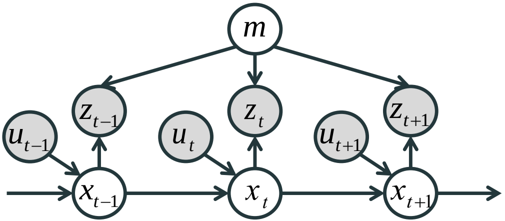
Use a **graphical model** to illustrate dependencies between robot position (x_t), position of landmarks (m) and control/sensory input (u_t, z_t).



The **observed variables** are shaded in the graphical model.

SLAM graphical model

Use a **graphical model** to illustrate dependencies between robot position (x_t), position of landmarks (m) and control/sensory input (u_t, z_t).



The **observed variables** are shaded in the graphical model.

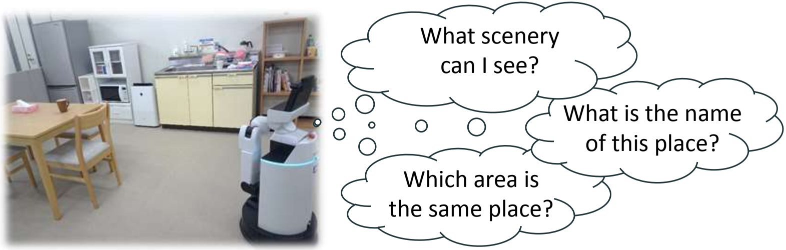
The robot's task: Compute the posterior distribution

$$p(\texttt{position}, \texttt{landmarks} \mid \texttt{data}) = p(x_t, m \mid u_{1:t}, z_{1:t}).$$

SLAM graphical model

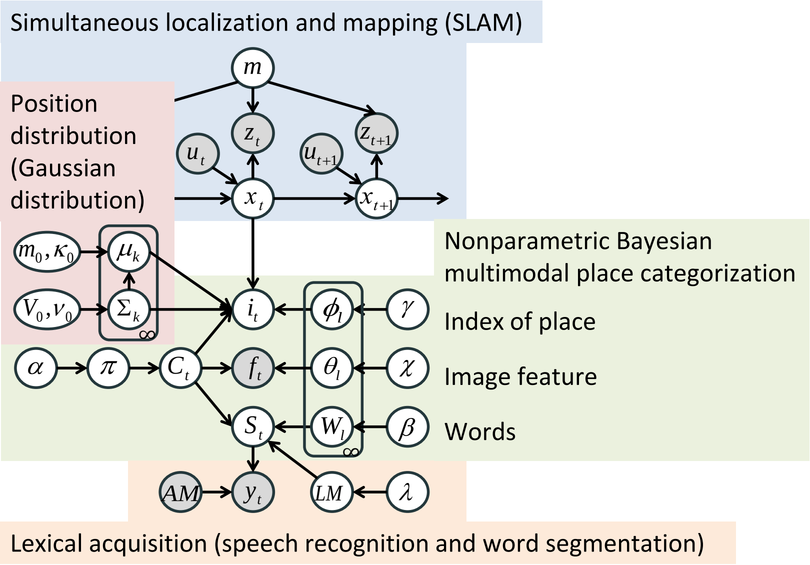
What scenery
can I see?

SLAM graphical model



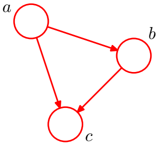
A. Taniguchi, Y. Hagiwara, T. Taniguchi, and T. Inamura. **Online Spatial Concept and Lexical Acquisition with Simultaneous Localization and Mapping.** *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

SLAM graphical model



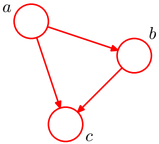
Graphical models

1. **Directed graphs** (a.k.a. Bayesian networks) represent a set of random variables and their conditional dependence structure.

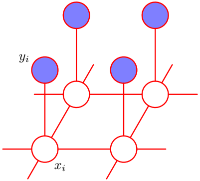


Graphical models

1. **Directed graphs** (a.k.a. Bayesian networks) represent a set of random variables and their conditional dependence structure.

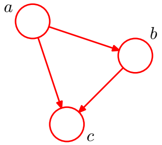


2. **Undirected graphs** (a.k.a. Markov random fields) represents a set of random variables and their Markov structure.

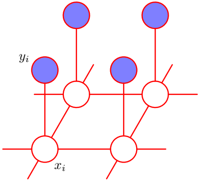


Graphical models

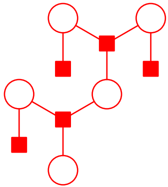
1. **Directed graphs** (a.k.a. Bayesian networks) represent a set of random variables and their conditional dependence structure.



2. **Undirected graphs** (a.k.a. Markov random fields) represents a set of random variables and their Markov structure.



3. **Factor graphs** make the factorization of the joint distribution of all variables more explicit.



Bayesian inference

In this module: Assume that the **graph** and the corresponding **conditional distributions** are given.

Task: The graphical model **inference problem** is to compute the Bayesian posterior distribution

$$p(\texttt{unobserved variables} \,|\, \texttt{observed variables}).$$

Bayesian inference

In this module: Assume that the **graph** and the corresponding **conditional distributions** are given.

Task: The graphical model **inference problem** is to compute the Bayesian posterior distribution

$$p(\texttt{unobserved variables} \mid \texttt{observed variables}).$$

Related tasks:

- Compute marginals $p(x_i \mid \texttt{observed variables})$.
- Compute the **marginal likelihood** $p(\texttt{observed variables})$.

What are the goals with this module?

Get a better understanding for how **probabilistic graphical models** can be used to model data dependencies in practical **applications**.

What are the goals with this module?

Get a better understanding for how **probabilistic graphical models** can be used to model data dependencies in practical **applications**.

Understand the graphical model inference problem and the need for **computational algorithms**.

What are the goals with this module?

Get a better understanding for how **probabilistic graphical models** can be used to model data dependencies in practical **applications**.

Understand the graphical model inference problem and the need for **computational algorithms**.

- Probabilistic ranking with TrueSkill *(today)*
- Topic modeling with Latent Dirichlet Allocation *(tomorrow)*

What are the goals with this module?

Get insight into popular **computational inference algorithms** for probabilistic graphical models:

- Sum–product algorithm, expectation propagation *(today)*
- Gibbs sampling, Markov chain Monte Carlo *(today/tomorrow)*
- (Stochastic) Variational inference *(tomorrow)*

What are the goals with this module?

Get insight into popular **computational inference algorithms** for probabilistic graphical models:

- Sum–product algorithm, expectation propagation *(today)*
- Gibbs sampling, Markov chain Monte Carlo *(today/tomorrow)*
- (Stochastic) Variational inference *(tomorrow)*

Extra (video lecture available on Canvas):

Get a feeling for how **probabilistic programming** can be used to automate inference and make probabilistic modeling more accessible to end users.

Who are we?

Fredrik Lindsten (LiU)

Johan Alenlöv (LiU)

David Broman (KTH)

Probabilistic ranking

Ranking systems

Competition is a central part of our society!

Ranking systems are widely used to rate teams & players, for the purpose of...

- ...matchmaking in online gaming,
- ...sports analytics,
- ...qualification to tournaments etc.

ex) WTA Ranking system

The Women's Tennis Association (WTA) singles rankings.

WTA rankings (singles) as of 16 October 2023[6]			
No.	Player	Points	Move
1	Aryna Sabalenka (BLR)	9,381	–
2	Iga Świątek (POL)	8,545	–
3	Coco Gauff (USA)	6,455	–
4	Jessica Pegula (USA)	5,985	–
5	Elena Rybakina (KAZ)	5,870	–
6	Maria Sakkari (GRE)	4,475	–
7	Ons Jabeur (TUN)	4,195	–
8	Markéta Vondroušová (CZE)	3,839	–
9	Karolína Muchová (CZE)	3,664	–
10	Caroline Garcia (FRA)	3,450	–
11	Daria Kasatkina (RUS)	2,880	▲ 1
12	Madison Keys (USA)	2,841	▼ 1
13	Barbora Krejčíková (CZE)	2,730	▲ 5
14	Jeļena Ostapenko (LAT)	2,665	▼ 1
15	Petra Kvitová (CZE)	2,660	▼ 1

https://en.wikipedia.org/wiki/WTA_Rankings

ex) WTA Ranking system

The WTA ranking system uses a (complex) set of rules to award points for different matches.

Category	W	F	SF	QF	R16	R32	R64	R128	Q	Q3	Q2	Q1
Grand Slam (S)	2000	1300	780	430	240	130	70	10	40	30	20	2
Grand Slam (D)	2000	1300	780	430	240	130	10	–	40	–	–	–
WTA Finals (S)	1500*	1080*	750*	(+125 per Round Robin Match; +125 per Round Robin Win)								
WTA Finals (D)	1500	1080	750	375	–							
WTA Premier Mandatory (96S)	1000	650	390	215	120	65	35	10	30	–	20	2
WTA Premier Mandatory (64/60S)	1000	650	390	215	120	65	10	–	30	–	20	2
WTA Premier Mandatory (28/32D)	1000	650	390	215	120	10	–	–	–	–	–	–
WTA Premier 5 (56S,64Q)	900	585	350	190	105	60	1	–	30	22	15	1
WTA Premier 5 (56S,48/32Q)	900	585	350	190	105	60	1	–	30	–	20	1
WTA Premier 5 (28D)	900	585	350	190	105	1	–	–	–	–	–	–
WTA Premier 5 (16D)	900	585	350	190	1	–	–	–	–	–	–	–
WTA Elite Trophy (S)	700*	440*	240*	(+40 per Round Robin Match; +80 per Round Robin Win)								
WTA Premier (56S)	470	305	185	100	55	30	1	–	25	–	13	1
WTA Premier (32S)	470	305	185	100	55	1	–	–	25	18	13	1
WTA Premier (16D)	470	305	185	100	1	–	–	–	–	–	–	–

ex) WTA Ranking system

The WTA ranking system uses a (complex) set of rules to award points for different matches.

Category	W	F	SF	QF	R16	R32	R64	R128	Q	Q3	Q2	Q1
Grand Slam (S)	2000	1300	780	430	240	130	70	10	40	30	20	2
Grand Slam (D)	2000	1300	780	430	240	130	10	–	40	–	–	–
WTA Finals (S)	1500*	1080*	750*	(+125 per Round Robin Match; +125 per Round Robin Win)								
WTA Finals (D)	1500	1080	750	375	–							
WTA Premier Mandatory (96S)	1000	650	390	215	120	65	35	10	30	–	20	2
WTA Premier Mandatory (64/60S)	1000	650	390	215	120	65	10	–	30	–	20	2
WTA Premier Mandatory (28/32D)	1000	650	390	215	120	10	–	–	–	–	–	–
WTA Premier 5 (56S,64Q)	900	585	350	190	105	60	1	–	30	22	15	1
WTA Premier 5 (56S,48/32Q)	900	585	350	190	105	60	1	–	30	–	20	1
WTA Premier 5 (28D)	900	585	350	190	105	1	–	–	–	–	–	–
WTA Premier 5 (16D)	900	585	350	190	1	–	–	–	–	–	–	–
WTA Elite Trophy (S)	700*	440*	240*	(+40 per Round Robin Match; +80 per Round Robin Win)								
WTA Premier (56S)	470	305	185	100	55	30	1	–	25	–	13	1
WTA Premier (32S)	470	305	185	100	55	1	–	–	25	18	13	1
WTA Premier (16D)	470	305	185	100	1	–	–	–	–	–	–	–

...but what does the points mean? How much should you **rationally** bet on Aryna Sabalenka vs Maria Sakkari?

A probabilistic ranking system

Instead of assigning points according to some set of rules, we will model each **player's skill** as a **latent random variable**.

$\Rightarrow$ probabilistic inference based on observed match outcomes.

A probabilistic ranking system

Instead of assigning points according to some set of rules, we will model each **player's skill** as a **latent random variable**.

$\Rightarrow$ probabilistic inference based on observed match outcomes.

Model: Let w_1 and w_2 denote the skills of players 1 and 2, respectively. A match **outcome** is modeled as:

1. Compute the skill difference $s = w_1 - w_2$.

A probabilistic ranking system

Instead of assigning points according to some set of rules, we will model each **player's skill** as a **latent random variable**.

$\Rightarrow$ probabilistic inference based on observed match outcomes.

Model: Let w_1 and w_2 denote the skills of players 1 and 2, respectively. A match **outcome** is modeled as:

1. Compute the skill difference $s = w_1 - w_2$.
2. Add "noise" to account for performance inconsistency/random effects,

$$t = s + v, \qquad\qquad v \sim N(0, 1).$$

A probabilistic ranking system

Instead of assigning points according to some set of rules, we will model each **player's skill** as a **latent random variable**.

$\Rightarrow$ probabilistic inference based on observed match outcomes.

Model: Let w_1 and w_2 denote the skills of players 1 and 2, respectively. A match **outcome** is modeled as:

1. Compute the skill difference $s = w_1 - w_2$.
2. Add "noise" to account for performance inconsistency/random effects,

$$t = s + v, \qquad\qquad v \sim N(0, 1).$$

3. Compute $y = \text{sign}(t)$,

$$\begin{cases} y = +1, & \text{player 1 wins,} \\ y = -1, & \text{player 2 wins.} \end{cases}$$

TrueSkill summary

Likelihood:

For any match between players i and j,

$$p(t \mid w_i, w_j) = N(t \mid w_i - w_j, 1),$$
$$p(y \mid t) = \delta_{\mathrm{sign}(t)}(y).$$

Skill prior:

For each player $i = 1, \ldots, M$,

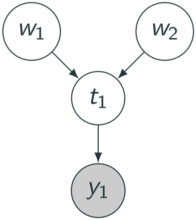
$$p(w_i) = N(w_i \mid 0, \sigma_0^2).$$

TrueSkill summary

Likelihood:

For any match between players i and j,

$$p(t \mid w_i, w_j) = N(t \mid w_i - w_j, 1),$$
$$p(y \mid t) = \delta_{\mathrm{sign}(t)}(y).$$



Skill prior:

For each player $i = 1, \ldots, M$,

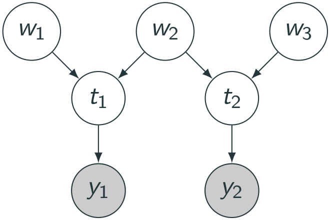
$$p(w_i) = N(w_i \mid 0, \sigma_0^2).$$

TrueSkill summary

Likelihood:

For any match between players i and j,

$$p(t \mid w_i, w_j) = N(t \mid w_i - w_j, 1),$$
$$p(y \mid t) = \delta_{\mathrm{sign}(t)}(y).$$



Skill prior:

For each player $i = 1, \ldots, M$,

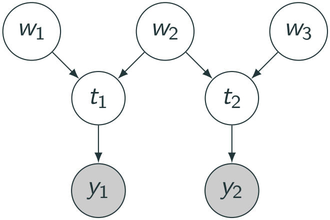
$$p(w_i) = N(w_i \mid 0, \sigma_0^2).$$

TrueSkill summary

Likelihood:

For any match between players i and j,

$$p(t \mid w_i, w_j) = N(t \mid w_i - w_j, 1),$$
$$p(y \mid t) = \delta_{\mathrm{sign}(t)}(y).$$



Skill prior:

For each player $i = 1, \dots, M$,

$$p(w_i) = N(w_i \mid 0, \sigma_0^2).$$

This is a **simplified version** of the TrueSkill™ ranking system,

R. Herbrich, T. Minka, and T. Graepel. **TrueSkill™: A Bayesian Skill Rating System.** *Advances in Neural Information Processing Systems 20*, 2007.

Bayesian inference

Task: Compute the posterior distribution

$$\pi(w_{1:3}, t_{1:2}) \overset{\text{def}}{=} p(w_{1:3}, t_{1:2} \mid y_{1:2}) \propto p(w_{1:3}, t_{1:2}, y_{1:2}).$$

Task: Compute the posterior distribution

$$\pi(w_{1:3}, t_{1:2}) \stackrel{\text{def}}{=} p(w_{1:3}, t_{1:2} \mid y_{1:2}) \propto p(w_{1:3}, t_{1:2}, y_{1:2}).$$

Always write down the joint PDF of everything!

$$\begin{aligned} p(w_{1:3}, t_{1:2}, y_{1:2}) = {} & p(w_1)p(w_2)p(w_3) \\ & \times N(t_1 \mid w_1 - w_2, 1)N(t_2 \mid w_2 - w_3, 1)\delta_{\mathrm{sign}(t_1)}(y_1)\delta_{\mathrm{sign}(t_2)}(y_2). \end{aligned}$$

Bayesian inference

Task: Compute the posterior distribution

$$\pi(w_{1:3}, t_{1:2}) \stackrel{\text{def}}{=} p(w_{1:3}, t_{1:2} \mid y_{1:2}) \propto p(w_{1:3}, t_{1:2}, y_{1:2}).$$

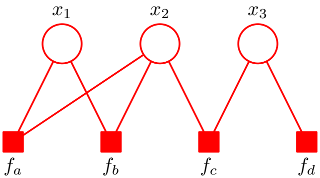
Always write down the joint PDF of everything!

$$\begin{aligned} p(w_{1:3}, t_{1:2}, y_{1:2}) = {} & p(w_1)p(w_2)p(w_3) \\ & \times N(t_1 \mid w_1 - w_2, 1)N(t_2 \mid w_2 - w_3, 1)\delta_{\mathrm{sign}(t_1)}(y_1)\delta_{\mathrm{sign}(t_2)}(y_2). \end{aligned}$$

The normalization constant $p(y_{1:2}) = \int p(w_{1:3}, t_{1:2}, y_{1:2}) dw_{1:3} dt_{1:2}$ is intractable.

Factor graphs and message passing

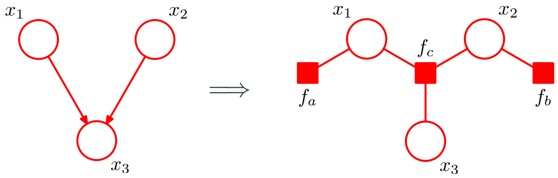
Factor graphs



$$\pi(x_{1:3}) = \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

- We can convert both directed and undirected graphical models to **factor graphs**.
- These have both **variable nodes** and **factor nodes**, which form a bipartite graph.
- The motivation is to make the factors more explicit, and facilitate inference algorithms based on **message passing**.

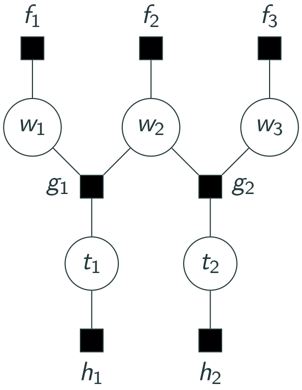
Directed graphical model $\Longrightarrow$ factor graph



$$\pi(x_{1:3}) = \underbrace{f_a(x_1)}_{=p(x_1)}\,\underbrace{f_b(x_2)}_{=p(x_2)}\,\underbrace{f_c(x_1, x_2, x_3)}_{=p(x_3 \mid x_1, x_2)}.$$

1. Create a variable node for each node in the original graph.
2. Create a factor node for each node in the original graph, where this factor expresses its conditional probability distribution.
3. Observed variables can either be kept as (observed) variable nodes, or be incorporated in the factors!

Factor graph for TrueSkill



Skill factors: For $i = 1, \dots, M$

$$f_i(w_i) \stackrel{\text{def}}{=} p(w_i) = N(w_i \mid 0, \sigma_0^2)$$

Game factors: For $k = 1, \dots, N$

$$\begin{aligned} g_k(t_k, w_{I_k}, w_{J_k}) &= p(t_k \mid w_{I_k}, w_{J_k}) \\ &= N(t_k \mid w_{I_k} - w_{J_k}, 1). \end{aligned}$$

(I_k and J_k are the players of game k).

Outcome factors: For $k = 1, \dots, N$

$$\begin{aligned} h_k(t_k) &= p(y_k \mid t_k) \\ &= \delta_{\text{sign}(t_k)}(y_k) = \mathbb{1}(y_k t_k > 0) \end{aligned}$$

Factor graph for TrueSkill

The **unnormalized PDF** over the variables of the factor graph is given by the product of all factors.

Let $\mathbf{w} = \{w_1, \ldots, w_M\}$, $\mathbf{t} = \{t_1, \ldots, t_N\}$, and $\mathbf{y} = \{y_1, \ldots, y_N\}$. Then

$$\pi(\mathbf{w}, \mathbf{t}) = \frac{1}{Z} \underbrace{\prod_{i=1}^{M} f_i(w_i) \prod_{k=1}^{N} g_k(t_k, w_{I_k}, w_{J_k}) h_k(t_k)}_{=p(\mathbf{w},\mathbf{t},\mathbf{y})}$$

Factor graph for TrueSkill

The **unnormalized PDF** over the variables of the factor graph is given by the product of all factors.

Let $\mathbf{w} = \{w_1, \ldots, w_M\}$, $\mathbf{t} = \{t_1, \ldots, t_N\}$, and $\mathbf{y} = \{y_1, \ldots, y_N\}$. Then

$$\pi(\mathbf{w}, \mathbf{t}) = \frac{1}{Z} \underbrace{\prod_{i=1}^{M} f_i(w_i) \prod_{k=1}^{N} g_k(t_k, w_{I_k}, w_{J_k}) h_k(t_k)}_{=p(\mathbf{w},\mathbf{t},\mathbf{y})}$$

Consequently...

- $Z = p(\mathbf{y})$ is the **marginal likelihood,**
- $\pi(\mathbf{w}, \mathbf{t}) = p(\mathbf{w}, \mathbf{t} \mid \mathbf{y})$ is the **posterior.**

Factor graphs

We will derive algorithms for **general factor graphs**.

- A factor graph is a triplet $(\mathcal{F}, \mathcal{V}, \mathcal{E})$, where $\mathcal{F}$ is the factor set, $\mathcal{V}$ is the variable set, and $\mathcal{E}$ is the edge set.

Factor graphs

We will derive algorithms for **general factor graphs**.

- A factor graph is a triplet $(\mathcal{F}, \mathcal{V}, \mathcal{E})$, where $\mathcal{F}$ is the factor set, $\mathcal{V}$ is the variable set, and $\mathcal{E}$ is the edge set.
- Note that $(s, i) \in \mathcal{E} \Rightarrow s \in \mathcal{F}$ and $i \in \mathcal{V}$.

Factor graphs

We will derive algorithms for **general factor graphs**.

- A factor graph is a triplet $(\mathcal{F}, \mathcal{V}, \mathcal{E})$, where $\mathcal{F}$ is the factor set, $\mathcal{V}$ is the variable set, and $\mathcal{E}$ is the edge set.
- Note that $(s, i) \in \mathcal{E} \Rightarrow s \in \mathcal{F}$ and $i \in \mathcal{V}$.
- We write

$$\begin{cases} \mathcal{N}(s) \subset \mathcal{V} & = \text{neighbors of factor } s, \\ \mathcal{N}(i) \subset \mathcal{F} & = \text{neighbors of variable } i, \end{cases}$$

Factor graphs

We will derive algorithms for **general factor graphs**.

- A factor graph is a triplet $(\mathcal{F}, \mathcal{V}, \mathcal{E})$, where $\mathcal{F}$ is the factor set, $\mathcal{V}$ is the variable set, and $\mathcal{E}$ is the edge set.
- Note that $(s, i) \in \mathcal{E} \Rightarrow s \in \mathcal{F}$ and $i \in \mathcal{V}$.
- We write

$$\begin{cases} \mathcal{N}(s) \subset \mathcal{V} & = \text{neighbors of factor } s, \\ \mathcal{N}(i) \subset \mathcal{F} & = \text{neighbors of variable } i, \end{cases}$$

- $\mathbf{x} = \{x_i : i \in \mathcal{V}\}$ is a shorthand for **all variables** of the model.

Factor graphs

We will derive algorithms for **general factor graphs**.

- A factor graph is a triplet $(\mathcal{F}, \mathcal{V}, \mathcal{E})$, where $\mathcal{F}$ is the factor set, $\mathcal{V}$ is the variable set, and $\mathcal{E}$ is the edge set.
- Note that $(s, i) \in \mathcal{E} \Rightarrow s \in \mathcal{F}$ and $i \in \mathcal{V}$.
- We write

$$\begin{cases} \mathcal{N}(s) \subset \mathcal{V} & = \text{neighbors of factor } s, \\ \mathcal{N}(i) \subset \mathcal{F} & = \text{neighbors of variable } i, \end{cases}$$

- $\mathbf{x} = \{x_i : i \in \mathcal{V}\}$ is a shorthand for **all variables** of the model.
- The joint distribution is

$$\pi(\mathbf{x}) = \frac{1}{Z} \prod_{s \in \mathcal{F}} f_s(\mathbf{x}_s)$$

where $\mathbf{x}_s = \{x_i : (s, i) \in \mathcal{E}\}$.

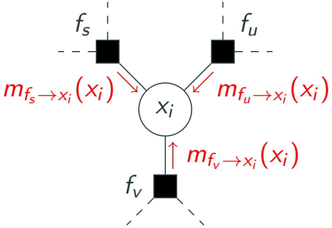
Sum–product algorithm

Recall: For a **tree-structured** graphical model the **marginal belief** at any node can be computed using **message passing**.

Specifically,

$$\pi(x_i) = \frac{1}{Z} \prod_{s \in \mathcal{N}(i)} m_{f_s \to x_i}(x_i)$$

where $m_{f_s \to x_i}(x_i)$ is the **message** from factor s to variable i.



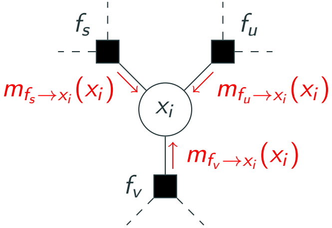
Sum–product algorithm

Recall: For a **tree-structured** graphical model the **marginal belief** at any node can be computed using **message passing**.

Specifically,

$$\pi(x_i) = \frac{1}{Z} \prod_{s \in \mathcal{N}(i)} m_{f_s \to x_i}(x_i)$$

where $m_{f_s \to x_i}(x_i)$ is the **message** from factor s to variable i.



Note: Same normalizing constant Z as in the expression for the joint $\pi(\mathbf{x})$ appears in all marginals, but this assumes that the messages are computed without (explicit or implicit) normalization!

Sum–product algorithm

1. Root the tree.
2. Initialize $m_{f_s \to x_i}(x_i) = f_s(x_i)$ if s is a leaf (factor) node, and $m_{x_i \to f_s}(x_i) = 1$ if i is a leaf (variable) node.

Sum–product algorithm

1. Root the tree.
2. Initialize $m_{f_s \to x_i}(x_i) = f_s(x_i)$ if s is a leaf (factor) node, and $m_{x_i \to f_s}(x_i) = 1$ if i is a leaf (variable) node.
3. Pass messages from leaves to root and back:

Factor-to-variable:

$$m_{f_s \to x_i}(x_i) = \int f_s(\mathbf{x}_s) \prod_{k \in \mathcal{N}(s) \setminus \{i\}} m_{x_k \to f_s}(x_k) d\{\mathbf{x}_s \setminus x_i\},$$

Sum–product algorithm

1. Root the tree.
2. Initialize $m_{f_s \to x_i}(x_i) = f_s(x_i)$ if s is a leaf (factor) node, and $m_{x_i \to f_s}(x_i) = 1$ if i is a leaf (variable) node.
3. Pass messages from leaves to root and back:

Factor-to-variable:

$$m_{f_s \to x_i}(x_i) = \int f_s(\mathbf{x}_s) \prod_{k \in \mathcal{N}(s) \setminus \{i\}} m_{x_k \to f_s}(x_k) d\{\mathbf{x}_s \setminus x_i\},$$

Variable-to-factor:

$$m_{x_i \to f_s}(x_i) = \prod_{u \in \mathcal{N}(i) \setminus \{s\}} m_{f_u \to x_i}(x_i).$$

Sum–product algorithm

1. Root the tree.
2. Initialize $m_{f_s \to x_i}(x_i) = f_s(x_i)$ if s is a leaf (factor) node, and $m_{x_i \to f_s}(x_i) = 1$ if i is a leaf (variable) node.
3. Pass messages from leaves to root and back:

   Factor-to-variable:

   $$m_{f_s \to x_i}(x_i) = \int f_s(\mathbf{x}_s) \prod_{k \in \mathcal{N}(s) \setminus \{i\}} m_{x_k \to f_s}(x_k) d\{\mathbf{x}_s \setminus x_i\},$$

   Variable-to-factor:

   $$m_{x_i \to f_s}(x_i) = \prod_{u \in \mathcal{N}(i) \setminus \{s\}} m_{f_u \to x_i}(x_i).$$

4. Compute marginals,

$$\pi(x_i) = \frac{1}{Z} \prod_{s \in \mathcal{N}(i)} m_{f_s \to x_i}(x_i)$$

Sum–product algorithm

Message passing:

- Messages are obtained by **partially** integrating out variables over **sub-trees**
- Computation (= integration) is **localized**

Sum–product algorithm

Message passing:

- Messages are obtained by **partially** integrating out variables over **sub-trees**
- Computation (= integration) is **localized**

Note that variable-to-factor messages can be expressed in terms of the marginals,

$$m_{x_i \to f_s}(x_i) \propto \frac{\pi(x_i)}{m_{f_s \to x_i}(x_i)}.$$

Will be useful later, when working with approximate marginals!

What do the messages actually look like?

The messages $m_{f_s \to x_i}(x_i)$ and $m_{x_i \to f_s}(x_i)$ are functions of the **model variables**. Each message has a set of hyperparameters that describe its functional form.

What do the messages actually look like?

The messages $m_{f_s \to x_i}(x_i)$ and $m_{x_i \to f_s}(x_i)$ are functions of the **model variables**. Each message has a set of hyperparameters that describe its functional form.

- **Discrete-valued variables:** the hyperparameters are the function values for each possible value of the input.

What do the messages actually look like?

The messages $m_{f_s \to x_i}(x_i)$ and $m_{x_i \to f_s}(x_i)$ are functions of the **model variables**. Each message has a set of hyperparameters that describe its functional form.

- **Discrete-valued variables:** the hyperparameters are the function values for each possible value of the input.
- **Continuous-valued variables:** the messages belong to some parametric family. E.g., for Gaussian models, each message is a (possibly unnormalized) Gaussian PDF, and the hyperparameters are its mean and variance.

Message passing in TrueSkill

In the TrueSkill model most factors are Gaussian. We use **Gaussian PDFs** to represent messages.

Message passing in TrueSkill

In the TrueSkill model most factors are Gaussian. We use **Gaussian PDFs** to represent messages.

A few useful facts about Gaussians:

$N(\mu, \sigma^2)$ can be parameterized on **information form** by,

$$\lambda = \frac{1}{\sigma^2} \quad \textit{(precision)} \qquad \text{and} \qquad \nu = \lambda\mu \quad \textit{(natural mean)}$$

We write $N_I(\nu, \lambda)$ for the PDF.

Message passing in TrueSkill

In the TrueSkill model most factors are Gaussian. We use **Gaussian PDFs** to represent messages.

A few useful facts about Gaussians:

$N(\mu, \sigma^2)$ can be parameterized on **information form** by,

$$\lambda = \frac{1}{\sigma^2} \quad \textit{(precision)} \qquad \text{and} \qquad \nu = \lambda\mu \quad \textit{(natural mean)}$$

We write $N_I(\nu, \lambda)$ for the PDF.

- Mean and variance are additive under convolution:

$$\int N(x|y, \gamma^2) N(y \mid \mu, \sigma^2) dy = N(x \mid \mu, \gamma^2 + \sigma^2).$$

Message passing in TrueSkill

In the TrueSkill model most factors are Gaussian. We use **Gaussian PDFs** to represent messages.

A few useful facts about Gaussians:

$N(\mu, \sigma^2)$ can be parameterized on **information form** by,

$$\lambda = \frac{1}{\sigma^2} \quad \textit{(precision)} \qquad \text{and} \qquad \nu = \lambda\mu \quad \textit{(natural mean)}$$

We write $N_I(\nu, \lambda)$ for the PDF.

- Mean and variance are additive under convolution:
$$\int N(x|y, \gamma^2) N(y \mid \mu, \sigma^2) dy = N(x \mid \mu, \gamma^2 + \sigma^2).$$
- Natural parameters are additive under multiplication:
$$N_I(x|\nu_1, \lambda_1) N_I(x \mid \nu_2, \lambda_2) \propto N_I(x \mid \nu_1 + \nu_2, \lambda_1 + \lambda_2).$$

Dealing with non-conjugacy

A first problem! The marginal belief $\pi(t)$ does not belong to to our parametric family of distributions (i.e., Gaussian).

The consecutive messages will lack **convenient parametric forms**, and the integrals involved will be **intractable**.

Dealing with non-conjugacy

A first problem! The marginal belief $\pi(t)$ does not belong to to our parametric family of distributions (i.e., Gaussian).

The consecutive messages will lack **convenient parametric forms**, and the integrals involved will be **intractable**.

Pragmatic solution: replace the problematic messages by Gaussian messages using **moment matching**,

$$\pi(t) \approx q(t) \stackrel{\text{def}}{=} N(t \mid \widetilde{\mu}, \widetilde{\sigma}^2)$$

where $\widetilde{\mu} = \int t\pi(t)dt$ and $\widetilde{\sigma}^2 = \int (t - \widetilde{\mu})^2 \pi(t)dt$.

Dealing with non-conjugacy

A first problem! The marginal belief $\pi(t)$ does not belong to to our parametric family of distributions (i.e., Gaussian).

The consecutive messages will lack **convenient parametric forms**, and the integrals involved will be **intractable**.

Pragmatic solution: replace the problematic messages by Gaussian messages using **moment matching**,

$$\pi(t) \approx q(t) \stackrel{\text{def}}{=} N(t \mid \widetilde{\mu}, \widetilde{\sigma}^2)$$

where $\widetilde{\mu} = \int t\pi(t)dt$ and $\widetilde{\sigma}^2 = \int (t - \widetilde{\mu})^2 \pi(t)dt$.

More generally, we project the messages onto some parameteric family of functions by Kullback–Leibler minimization.

Approximate message passing

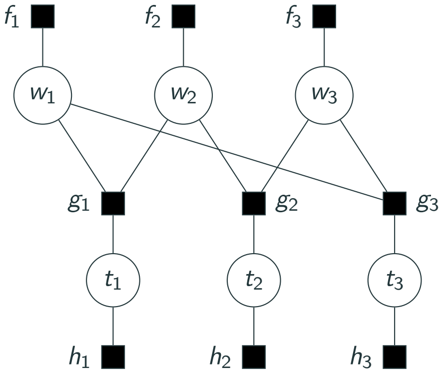
We get a practical algorithm!

Approximate message passing:

1. Root the tree
2. Propagate messages from leafs to root and back.
   - For non-conjugate messages, **approximate the marginal belief** by a parametric distribution and compute the message accordingly.
3. Compute/report marginal beliefs at all nodes of interest.

Dealing with loopy graphs

What if another match is played between player 1 and player 3?



A second problem! The graph is no longer a tree!

Dealing with loopy graphs

Pragmatic solution: Pretend that this is not a problem...

Loopy message–passing:

1. Initialize the messages arbitrarily
2. Keep propagating messages until convergence

Dealing with loopy graphs

Pragmatic solution: Pretend that this is not a problem...

Loopy message–passing:

1. Initialize the messages arbitrarily
2. Keep propagating messages until convergence

No convergence guarantees!

Dealing with loopy graphs

Pragmatic solution: Pretend that this is not a problem...

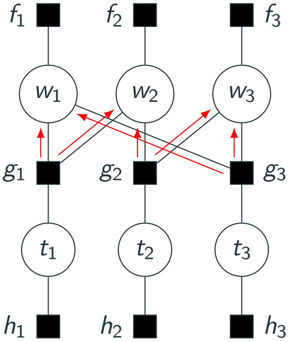
Loopy message–passing:

1. Initialize the messages arbitrarily
2. Keep propagating messages until convergence

No convergence guarantees!

- Stationary points can be characterized
- Many extensions, e.g. based on message-tempering
- Often works well in practice

Expectation propagation for TrueSkill



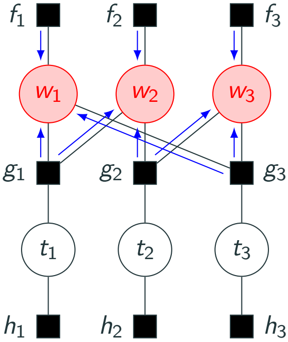
Step 0: Initialize game-to-skill messages

$$m^{(0)}_{g_k \to w_{I_k}}(w_{I_k}) \equiv 1,$$
$$m^{(0)}_{g_k \to w_{J_k}}(w_{J_k}) \equiv 1,$$

for $k = 1, \ldots, N$. Set $\tau = 1$.

($N \times 2$ messages)

Expectation propagation for TrueSkill



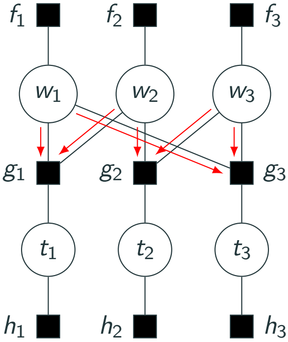
Step 1: Update skill marginals

$$q^{(\tau)}(w_i) \propto f_i(w_i) \prod_{\substack{k=1,\ldots,N \\ \text{s.t. } i \in \{I_k, J_k\}}} m^{(\tau-1)}_{g_k \to w_i}(w_i)$$

for $i = 1, \ldots, M$.

(M marginals)

Expectation propagation for TrueSkill



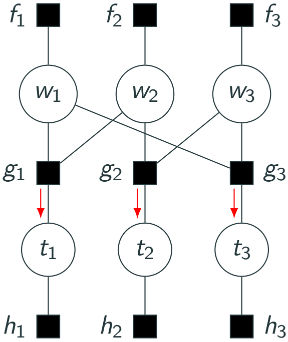
Step 2: Compute skill-to-game messages

$$m^{(\tau)}_{w_i \to g_k}(w_i) = \frac{q^{(\tau)}(w_i)}{m^{(\tau-1)}_{g_k \to w_i}(w_i)}$$

for $k = 1, \ldots, N$ and $i \in \{I_k, J_k\}$.

($N \times 2$ messages)

Expectation propagation for TrueSkill



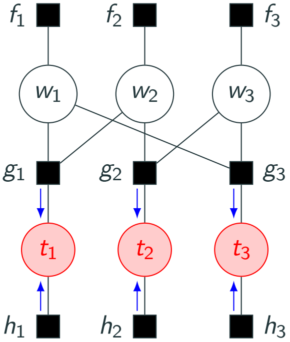
Step 3: Compute game-to-performance messages

$$m^{(\tau)}_{g_k \to t_k}(t_k) = \int g_k(t_k, w_{I_k}, w_{J_k}) \times m^{(\tau)}_{w_{I_k} \to g_k}(w_{I_k}) m^{(\tau)}_{w_{J_k} \to g_k}(w_{J_k}) dw_{I_k} w_{J_k}$$

for $k = 1, \ldots, N$.

(N messages)

Expectation propagation for TrueSkill



Step 4: Update performance marginals

$$\widehat{\pi}^{(\tau)}(t_k) \propto h_k(t_k) m^{(\tau)}_{g_k \to t_k}(t_k)$$

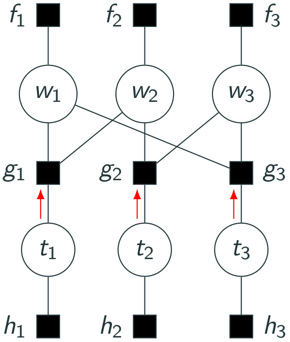
and approximate using moment matching,

$$q^{(\tau)}(t_k) \approx \widehat{\pi}^{(\tau)}(t_k)$$

for $k = 1, \ldots, N$.

(N marginals)

Expectation propagation for TrueSkill



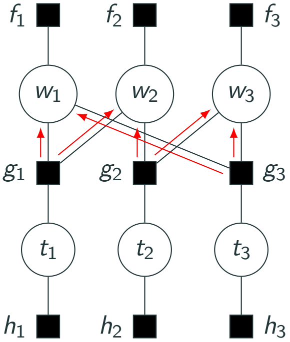
Step 5: Compute performance-to-game messages

$$m^{(\tau)}_{t_k \to g_k}(t_k) = \frac{q^{(\tau)}(t_k)}{m^{(\tau)}_{g_k \to t_k}(t_k)}$$

for $k = 1, \ldots, N$.

(N messages)

Expectation propagation for TrueSkill



Step 6: Compute game-to-skill messages

$$m^{(\tau)}_{g_k \to w_{I_k}}(w_{I_k}) = \int g_k(t_k, w_{I_k}, w_{J_k}) \times m^{(\tau)}_{t_k \to g_k}(t_k) m^{(\tau)}_{w_{J_k} \to g_k}(w_{J_k}) dt_k d_{w_{J_k}},$$

and similarly for $m^{(\tau)}_{g_k \to w_{J_k}}(w_{J_k})$, for $k = 1, \ldots, N$. Set $\tau \leftarrow \tau + 1$ and go back to step 1.

($N \times 2$ messages)

Expectation propagation

Assumed density filtering

This algorithm is a special case of **Expectation Propagation (EP)**

To derive EP, it is instructive to start with assumed density filtering.

ex, ADF) Let $\mathbf{y} = \{y_1, \ldots, y_n\}$ be a sequence of data points. We seek

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{x}) \prod_{i=1}^{n} p(y_i \mid \mathbf{x}).$$

In ADF we initialize $q^{(0)}(\mathbf{x}) = p(\mathbf{x})$ and then loop:

for $i = 1, \ldots, n$

- Compute $\widehat{p}(\mathbf{x} \mid y_{1:i}) \propto p(y_i \mid \mathbf{x}) q^{(i-1)}(\mathbf{x})$
- Compute $q^{(i)}(\mathbf{x}) = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}(\widehat{p} \| q)$

(Here $\mathcal{Q}$ denotes some appropriate class of distributions, e.g., in the exponential family.)

From ADF to EP

- In ADF we only see each data point once.
- If we keep iterating, the data will be double-counted.

From ADF to EP

- In ADF we only see each data point once.
- If we keep iterating, the data will be double-counted.

ex, EP) Approximate $p(\mathbf{x}|\mathbf{y})$ with $q(\mathbf{x}) \propto p(\mathbf{x}) \prod_{i=1}^{n} \tilde{f}_i(\mathbf{x})$, where $\tilde{f}_i(\mathbf{x})$ are of some "simple form", so that $q(\mathbf{x})$ is tractable.

for $i = 1, \ldots, n, 1, \ldots, n, \ldots,$ **until convergence**

- Compute $q^{(-i)}(\mathbf{x}) \propto q(\mathbf{x})/\tilde{f}_i(\mathbf{x})$
- Compute $\widehat{p}(\mathbf{x} \,|\, \mathbf{y}) \propto p(y_i \,|\, \mathbf{x}) q^{(-i)}(\mathbf{x})$
- Update $q(\mathbf{x}) \leftarrow \arg\min_{q \in \mathcal{Q}} \mathrm{KL}(\widehat{p} \| q)$
- Update $\tilde{f}_i(\mathbf{x}) \propto q(\mathbf{x})/q^{(-i)}(\mathbf{x})$

Expectation propagation

More generally: Assume that the target distribution factorizes

$$\pi(\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{n} f_i(\mathbf{x}).$$

We approximate $\pi(\mathbf{x}) \approx q(\mathbf{x})$ where

$$q(\mathbf{x}) \propto \prod_{i=1}^{n} \tilde{f}_i(\mathbf{x})$$

where the factors $\tilde{f}_i(\mathbf{x})$ are of some "simple form", so that $q(\mathbf{x})$ is tractable.

Expectation propagation

Initialize factors $\tilde{f}_i(\mathbf{x}) \Rightarrow q(\mathbf{x}) \propto \prod_{i=1}^n \tilde{f}_i(\mathbf{x})$.

while not converged

1. Pick a factor i to update.
2. Compute the **cavity distribution:** $q^{(-i)}(\mathbf{x}) \propto q(\mathbf{x})/\tilde{f}_i(\mathbf{x})$.
3. Compute the **tilted distribution:** $\widehat{\pi}(\mathbf{x}) = f_i(\mathbf{x})q^{(-i)}(\mathbf{x})/Z_i$ where $Z_i = \int f_i(\mathbf{x})q^{(-i)}(\mathbf{x})d\mathbf{x}$.
4. Update the approximation: $q(\mathbf{x}) \leftarrow \arg\min_{q\in\mathcal{Q}} \mathrm{KL}(\widehat{\pi}\|q)$.
5. Update the ith factor: $\tilde{f}_i(\mathbf{x}) \leftarrow Z_i q(\mathbf{x})/q^{(-i)}(\mathbf{x})$.

Expectation propagation

Initialize factors $\tilde{f}_i(\mathbf{x}) \Rightarrow q(\mathbf{x}) \propto \prod_{i=1}^{n} \tilde{f}_i(\mathbf{x})$.

while not converged

1. Pick a factor i to update.
2. Compute the **cavity distribution:** $q^{(-i)}(\mathbf{x}) \propto q(\mathbf{x})/\tilde{f}_i(\mathbf{x})$.
3. Compute the **tilted distribution:** $\widehat{\pi}(\mathbf{x}) = f_i(\mathbf{x})q^{(-i)}(\mathbf{x})/Z_i$ where $Z_i = \int f_i(\mathbf{x})q^{(-i)}(\mathbf{x})d\mathbf{x}$.
4. Update the approximation: $q(\mathbf{x}) \leftarrow \arg\min_{q\in\mathcal{Q}} \mathrm{KL}(\widehat{\pi}\|q)$.
5. Update the ith factor: $\tilde{f}_i(\mathbf{x}) \leftarrow Z_i q(\mathbf{x})/q^{(-i)}(\mathbf{x})$.

Note: incorporating the constant Z_i in the updated factor in step 5 allows us to approximate the normalizing constant of $\pi(\mathbf{x})$ by

$$Z \approx \int \prod_{i=1}^{n} \tilde{f}_i(\mathbf{x})d\mathbf{x}.$$

Message passing as EP

Consider a **factor graph**

$$\pi(\mathbf{x}) = \frac{1}{Z} \prod_{s \in \mathcal{F}} f_s(\mathbf{x}_s).$$

Message passing as EP

Consider a **factor graph**

$$\pi(\mathbf{x}) = \frac{1}{Z} \prod_{s \in \mathcal{F}} f_s(\mathbf{x}_s).$$

To apply EP, we select an approximating distribution which factorizes over the model variables,

$$q(\mathbf{x}) \propto \prod_{s \in \mathcal{F}} \tilde{f}_s(\mathbf{x}_s) = \prod_{s \in \mathcal{F}} \prod_{k \in \mathcal{N}(s)} \tilde{t}_{sk}(x_k)$$

Message passing as EP

Consider a **factor graph**

$$\pi(\mathbf{x}) = \frac{1}{Z} \prod_{s \in \mathcal{F}} f_s(\mathbf{x}_s).$$

To apply EP, we select an approximating distribution which factorizes over the model variables,

$$q(\mathbf{x}) \propto \prod_{s \in \mathcal{F}} \tilde{f}_s(\mathbf{x}_s) = \prod_{s \in \mathcal{F}} \prod_{k \in \mathcal{N}(s)} \tilde{t}_{sk}(x_k)$$

The marginal of variable x_k is given by

$$q(x_k) \propto \prod_{s \in \mathcal{N}(k)} \tilde{t}_{sk}(x_k)$$

and we can identify $\tilde{t}_{sk}(x_k) \propto m_{f_s \to x_k}(x_k)$.

Message passing as EP

Consider a **factor graph**

$$\pi(\mathbf{x}) = \frac{1}{Z} \prod_{s \in \mathcal{F}} f_s(\mathbf{x}_s).$$

To apply EP, we select an approximating distribution which factorizes over the model variables,

$$q(\mathbf{x}) \propto \prod_{s \in \mathcal{F}} \tilde{f}_s(\mathbf{x}_s) = \prod_{s \in \mathcal{F}} \prod_{k \in \mathcal{N}(s)} \tilde{t}_{sk}(x_k)$$

The marginal of variable x_k is given by

$$q(x_k) \propto \prod_{s \in \mathcal{N}(k)} \tilde{t}_{sk}(x_k)$$

and we can identify $\tilde{t}_{sk}(x_k) \propto m_{f_s \to x_k}(x_k)$.

The EP algorithm, for this choice of $q(\mathbf{x})$, is equivalent to the (loopy and approximate) message passing method!