Factor graphs, belief propagation and variational inference

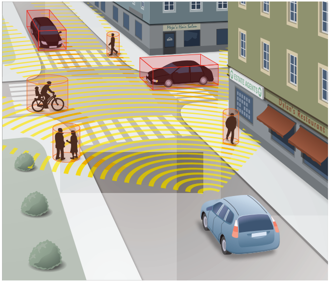
Lennart Svensson

Signal Processing Group
Department of Signals and Systems
Chalmers University of Technology, Sweden

Probabilistic graphical models (PGMs)

- PGMs are important in a wide range of applications:

- Self-driving vehicles.
- Cooperative localisation.
- Speech processing.
- Communication systems.
- Image segmentation.
- etc.



- **Purpose with PGMs**:
  - illustrate problem structure (conditional independencies),
  - exploit structure to design tractable inference algorithms.
- **Today's focus**: factor graphs, belief propagation and variational inference.

Why factor graphs?

Factor graphs are important/useful because:

- they provide new perspectives on "old" algorithms,
  $\rightsquigarrow$ filtering, smoothing, dynamic programming, Viterbi decoding, . . .
  are all **instances of factor graph algorithms**!
- they **visualize the structure of the problem**:
  clarify "dependencies" and how we can split one complicated function of many variables into simple functions.
- there are **efficient standard algorithms** that we can use, once we have defined the factor graph!

Learning objectives

After this lecture you should be able to

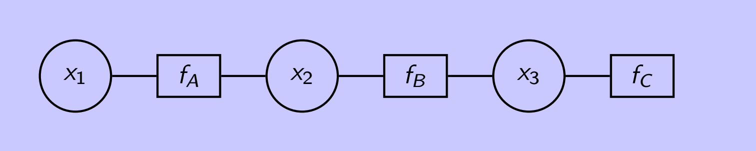
- formulate a factor graph (FG) given a factorization of a function,
- explain why it is important to make use of the structure/sparseness of a problem,
- describe how the sum-product algorithm works on a factor graph (without loops),
- summarize the basic ideas behind variational Bayes.

What is a factor graph?

- **Example 1:** given a factorization

$$g(x_1, x_2, x_3) = f_A(x_1, x_2) f_B(x_2, x_3) f_C(x_3)$$

we obtain the factor graph:



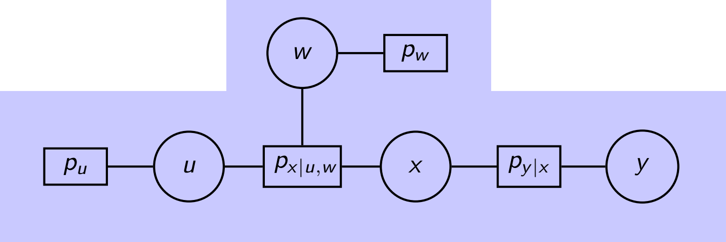
- A factor graph contains:
  - one **variable node** for each variable,
  - one **factor node** for each function,
  - one **edge** between x_i and f_j if x_i is a variable in f_j.

What is a factor graph?

- **Example 2:** given a factorization

$$p(u, w, x, y) = p_u(u)p_w(w)p_{x|u,w}(x \big| u, w)p_{y|x}(y \big| x)$$

we obtain the factor graph:



- A FG is a **bipartite** graph:
  - it contains two types of nodes,
  - edges always connect nodes of different types.
- Functions do not have to be probability densities.

What is a factor graph?

- **A DIY example:** given a probability density function

$$p(x, y, z) = p_x(x) p_{y|x}(y \big| x) p_{z|x,y}(z \big| x, y),$$

we obtain the factor graph:

Optional: for comparison you can also draw the corresponding Bayesian network.

Two important problems

Marginal distributions

- Find

$$p(x_i|\mathbf{y}) = \sum_{\sim x_i} p(\mathbf{x}|\mathbf{y})$$

  where $\sim x_i$ means "over all variables but x_i".

- **Example:** find $p(\mathbf{x}_k|\mathbf{y}_{1:k})$ from $p(\mathbf{x}_{1:k}|\mathbf{y}_{1:k})$, i.e., perform filtering.

Maximization

- Find

$$\hat{x}_i = \arg\max_{x_i} \max_{\sim x_i} p(\mathbf{x}|\mathbf{y})$$

- **Example:** find the most probable symbol in a communication message. Often closely related to dynamic programming.

Efficient marginalization

- **Example:** consider a function
$$g(x_1, x_2, x_3) = f_A(x_1) f_B(x_1, x_2) f_C(x_2, x_3)$$
where $x_1, x_2, x_3 \in \{1, 2, \ldots, N\}$. Describe how to compute
$$g_1(x_1) = \sum_{x_2, x_3} g(x_1, x_2, x_3)$$
efficiently.
Solution: the trick is to "push in the summations":
$$\sum_{x_2, x_3} g(x_1, x_2, x_3) = f_A(x_1) \sum_{x_2} f_B(x_1, x_2) \underbrace{\sum_{x_3} f_C(x_2, x_3)}_{\mu_{f_C \to x_2}(x_2)}$$
$$= f_A(x_1) \underbrace{\sum_{x_2} f_B(x_1, x_2) \mu_{f_C \to x_2}(x_2)}_{\mu_{f_B \to x_1}(x_1)}$$
$$\implies g_1(x_1) = f_A(x_1) \mu_{f_B \to x_1}(x_1)$$

Efficient marginalization

- **Example, DIY:** describe how to compute

$$g_2(x_2) = \sum_{x_1, x_3} f_A(x_1) f_B(x_1, x_2) f_C(x_2, x_3)$$

efficiently! (Remember to "push in" the summations.)

Why is structure important?

- Suppose
$$g(x_1, x_2, \ldots, x_k) = f_2(x_1, x_2) f_3(x_2, x_3) \ldots f_k(x_{k-1}, x_k),$$
where $x_1, x_2, \ldots, x_k \in \{1, 2, \ldots, N\}$.
- How many calculations are needed to compute
$$g_1(x_1) = \sum_{\sim x_1} g(x_1, x_2, \ldots, x_k)?$$
  - **Without using structure**: one summation over $k-1$ variables
$\Rightarrow \quad N^{k-1}$ terms for each value of x_1, i.e., $O(N^k)$ calculations.
  - **Pushing in summations:** $k-1$ summations over 1 variable
$\Rightarrow \quad O(k \times N^2)$ calculations.
- **Example:** $k = 100$ and $N = 2 \Rightarrow N^k \approx 1.3 \times 10^{30}$ and $k \times N^2 = 400$,
$\rightsquigarrow$ using the structure makes a **massive difference**!

The sum-product algorithm

The sum-product algorithm

- is also known as belief propagation,
- computes marginal distributions by “pushing in summations”,
- performs message passing on a graph,
- is exact for linear graphs and trees, but often performs remarkably well on general graphs (with loops),

The sum-product algorithm

The sum-product update rule

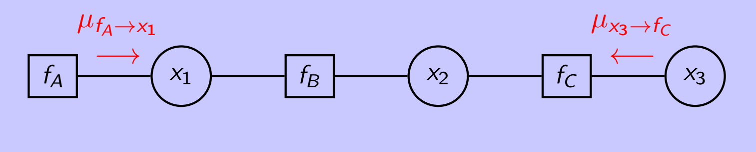
The message sent from a node v on an edge e is the product of the local function at v (or the unit function if v is a variable node) with all messages received at v on edges *other* than e, summarized for the variables not associated with e.

The sum-product algorithm

- Calculating marginal distributions of
$$g(x_1, x_2, x_3) = f_A(x_1) f_B(x_1, x_2) f_C(x_2, x_3)$$
- The sum-product algorithm operates in **three phases:**

Phase 1: initialization

- Send messages from the edges of the graph
  - messages from factor to variable: $\mu_{f_A \to x_1}(x_1) = f_A(x_1)$,
  - messages from variable to factor: $\mu_{x_3 \to f_C}(x_3) = 1$.

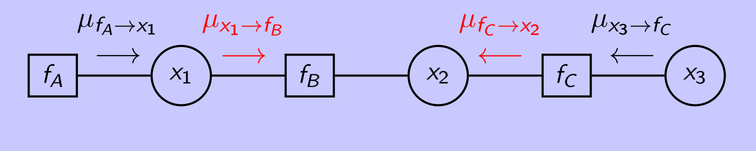


The sum-product algorithm

- Calculating marginal distributions of
$$g(x_1, x_2, x_3) = f_A(x_1) f_B(x_1, x_2) f_C(x_2, x_3)$$

Phase 2: message passing

- Compute outgoing messages when incoming message(s) are available:
  - messages from variable to factor: product of all incoming messages, $\mu_{x_1 \to f_B}(x_1) = \mu_{f_A \to x_1}(x_1)$,
  - messages from factor to variable: product of incoming messages and factor, sum out previous variables: $\mu_{f_C \to x_2}(x_2) = \sum_{x_3} \mu_{x_3 \to f_C}(x_3) f_C(x_2, x_3)$.



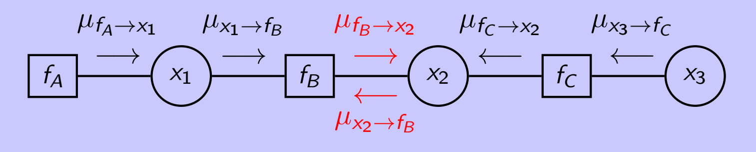
The sum-product algorithm

- Calculating marginal distributions of

$$g(x_1, x_2, x_3) = f_A(x_1) f_B(x_1, x_2) f_C(x_2, x_3)$$

Phase 2: message passing

- Compute outgoing messages when incoming message(s) are available:
  - $\mu_{f_B \to x_2}(x_2) = \sum_{x_1} \mu_{x_1 \to f_B}(x_1) f_B(x_1, x_2)$
  - $\mu_{x_2 \to f_B}(x_2) = \mu_{f_C \to x_2}(x_2)$



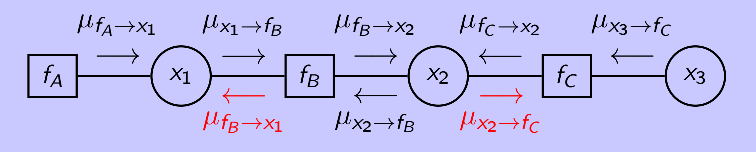
The sum-product algorithm

- Calculating marginal distributions of

$$g(x_1, x_2, x_3) = f_A(x_1) f_B(x_1, x_2) f_C(x_2, x_3)$$

Phase 2: message passing

- Compute outgoing messages when incoming message(s) are available:
  - $\mu_{f_B \to x_1}(x_1) = \sum_{x_2} \mu_{x_2 \to f_B}(x_2) f_B(x_1, x_2)$
  - $\mu_{x_2 \to f_C}(x_2) = \mu_{f_B \to x_2}(x_2)$



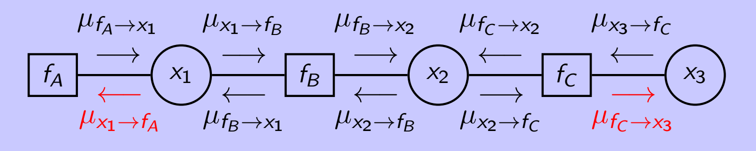
The sum-product algorithm

- Calculating marginal distributions of

$$g(x_1, x_2, x_3) = f_A(x_1) f_B(x_1, x_2) f_C(x_2, x_3)$$

Phase 2: message passing

- Compute outgoing messages when incoming message(s) are available:
  - $\mu_{x_1 \to f_A}(x_1) = \mu_{f_B \to x_1}(x_1)$
  - $\mu_{f_C \to x_3}(x_3) = \sum_{x_2} \mu_{x_2 \to f_C}(x_2) f_C(x_2, x_3)$



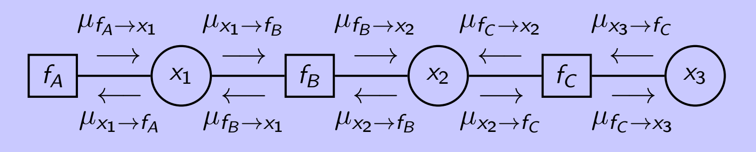
The sum-product algorithm

- Calculating marginal distributions of

$$g(x_1, x_2, x_3) = f_A(x_1) f_B(x_1, x_2) f_C(x_2, x_3)$$

Phase 3: termination

- A marginal distribution is the product of the incoming messages to the variable node:
  - $g_1(x_1) = \sum_{x_2, x_3} g(x_1, x_2, x_3) = \mu_{f_A \to x_1}(x_1) \mu_{f_B \to x_1}(x_1)$
  - $g_2(x_2) = \sum_{x_1, x_3} g(x_1, x_2, x_3) = \mu_{f_B \to x_2}(x_2) \mu_{f_C \to x_2}(x_2)$
  - $g_3(x_3) = \sum_{x_1, x_2} g(x_1, x_2, x_3) = \mu_{f_C \to x_3}(x_3)$.

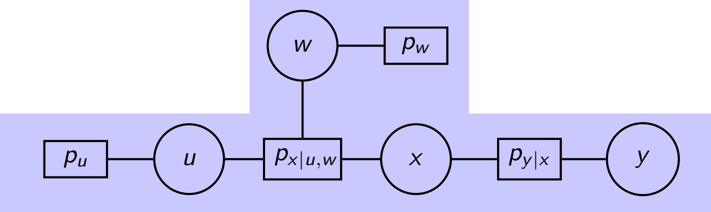


The sum-product algorithm

- **DIY:** verify that the sum-product algorithm computes $g_2(x_2)$ correctly.

Remarks on the sum-product algorithm

- We considered a linear graph, but the sum-product algorithm (SPA) is exact also for trees, like



- You get, e.g.,

$$\mu_{p_{x|u,w}\to x}(x) = \sum_{u,w} \mu_{w\to p_{x|u,w}}(w)\mu_{u\to p_{x|u,w}}(u)p_{x|u,w}(x|u,w)$$

- If the variables are continuous you replace summations with integrals.

Factor graphs and maximization

- We can find

$$\max_{\sim x_i} p(\mathbf{x} \mid \mathbf{y})$$

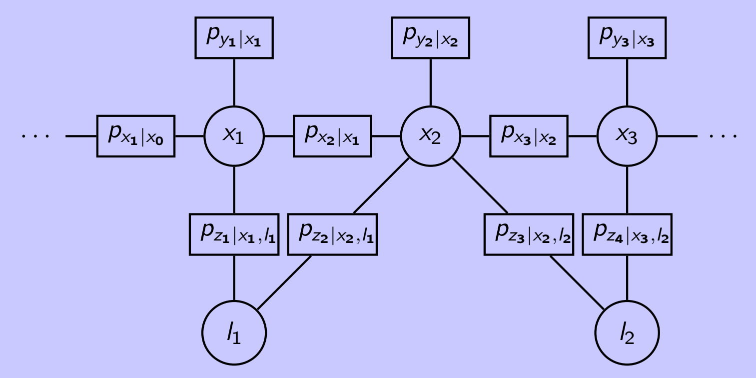
  using the **max-product algorithm**.

- The max-product algorithm is identical to the sum-product algorithm, but summations are "replaced by maximisations".

- For linear graphs, the max-product algorithm gives a version of **dynamic programming and Viterbi decoding**.

Factor graphs for SLAM

- For many problems, the factor graph contains loops.
- **Simultaneous localization and mapping (SLAM)**: position both a moving vehicle, $\mathbf{x}_1, \mathbf{x}_2, \ldots$, and different stationary landmarks, $l_1, l_2, \ldots$.



Graphs with loops

- Two important strategies for graphs with loops:

  1. **Use belief propagation** (the sum-product algorithm). The algorithm is no longer exact and needs to be iterated, but often yields remarkably good performance in practice.

  2. **Exact marginalization.** Often still a feasible alternative, but it is important to marginalize the variables in the correct order. We can still use the structure of the problem!

A variational perspective on BP, and more

- How can we approximate $p(\mathbf{x})$ when exact inference is intractable?

The variational idea

Find a tractable distribution $q(\mathbf{x}) \in \mathbf{Q}$ which is close to $p(\mathbf{x})$:

$$q(\mathbf{x}) = \arg \min_{\tilde{q}(\mathbf{x}) \in \mathbf{Q}} \mathrm{D}(\tilde{q}(\mathbf{x}) \| p(\mathbf{x})),$$

where $D(\tilde{q}(\mathbf{x}) \| p(\mathbf{x}))$ is small when $\tilde{q} \approx p$.

- By modifying $\mathbf{Q}$ and/or D we can derive belief propagation (BP), expectation propagation (EP), variational Bayes (VB), TRW-BP, GBP, Power-EP, etc.

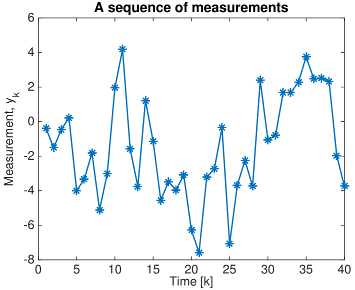
Motivating examples – estimation in SSMs

Let us study VB using a toy example:

- Consider a state space model

$$x_k = x_{k-1} + q_k, \quad q_k \sim \mathcal{N}(0, \tau_q^{-1})$$
$$y_k = x_k + r_k, \quad r_k \sim \mathcal{N}(0, \tau_r^{-1}).$$

- Can we estimate τ_q and τ_r from $y_1, \ldots, y_T$?
- **Difficulty:** the state sequence $x_1, \ldots, x_T$ is unknown!



Relevant?

- Enables us to estimate parameters without knowing the true state sequence.
- In practice, models tend to be nonlinear and high-dimensional which makes the problem less trivial.

Intractable problems?

- Let $\boldsymbol{\theta}$ denote parameters of interest, $\mathbf{y}$ be our observations and $\mathbf{x}$ represent the hidden variables.
  - In **toy example**: $\boldsymbol{\theta}$ contains mean and covariances, $\mathbf{y}$ are the sampels and $\mathbf{x}$ denotes assignments: measurements $\leftrightarrow$ Gaussian components.
- Can we compute $p(\boldsymbol{\theta}|\mathbf{y})$?
- An important complication is that we need $\mathbf{x}$ to express the relation between $\mathbf{y}$ and $\boldsymbol{\theta}$:

$$p(\mathbf{y}|\boldsymbol{\theta}) = \sum_{\mathbf{x}} p(\mathbf{y}, \mathbf{x}|\boldsymbol{\theta}),$$

which is often intractable.

Variational Bayesian theory

- **Idea 1:** find a distribution $q(\boldsymbol{\theta}, \mathbf{x})$ that approximates $p(\boldsymbol{\theta}, \mathbf{x} \big| \mathbf{y})$ well, in the sense that the Kullback-Leibler divergence (KLD)

$$\int q(\boldsymbol{\theta}, \mathbf{x}) \log \frac{q(\boldsymbol{\theta}, \mathbf{x})}{p(\boldsymbol{\theta}, \mathbf{x} \big| \mathbf{y})} \, d\boldsymbol{\theta} d\mathbf{x}$$

  is small.

- If $q(\boldsymbol{\theta}, \mathbf{x})$ has suitable properties, we can then easily find an approximation to $p(\boldsymbol{\theta} \big| \mathbf{y})$.

- **Note:** the optimal approximation in the KLD sense is $q(\boldsymbol{\theta}, \mathbf{x}) = p(\boldsymbol{\theta}, \mathbf{x} \big| \mathbf{y})$, but this is not tractable.
  $\rightsquigarrow$ We need to restrict $q(\boldsymbol{\theta}, \mathbf{x})$ to obtain a tractable solution!

Variational Bayesian theory

- **Idea 2:** seek the best approximation $q(\boldsymbol{\theta}, \mathbf{x}) \approx p(\boldsymbol{\theta}, \mathbf{x} | \mathbf{y})$ among all distributions that factorise $q(\boldsymbol{\theta}, \mathbf{x}) = q_\theta(\boldsymbol{\theta}) q_x(\mathbf{x})$.

Variational Bayesian – main results

Given $q_\theta(\boldsymbol{\theta})$, the optimal distribution $q_x(\mathbf{x})$ is

$$q_x(\mathbf{x}) \propto \exp\left(\mathbb{E}_{q_\theta(\boldsymbol{\theta})}\left[\log\left[p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})\right]\right]\right).$$

Given $q_x(\mathbf{x})$, the optimal distribution $q_\theta(\boldsymbol{\theta})$ is

$$q_\theta(\boldsymbol{\theta}) \propto \exp\left(\mathbb{E}_{q_x(\mathbf{x})}\left[\log\left[p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})\right]\right]\right).$$

- A few remarks:
  - We use these results to iteratively minimize the KLD.
  - We handle the distribution of the parameters of interests $\boldsymbol{\theta}$ and the hidden variables $\mathbf{x}$ in the same way.
  - We take expected values of $\log\left[p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})\right]$ instead of $p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})$, which **simplifies things considerably**.

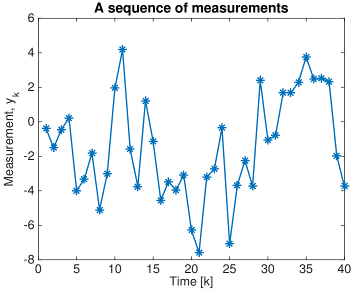
Example – VB solution (1)

Let us use VB to estimate parameters in a state space model.

- We have a state space model

$$x_k = x_{k-1} + q_k, \quad q_k \sim \mathcal{N}(0, \tau_q^{-1})$$
$$y_k = x_k + r_k, \quad r_k \sim \mathcal{N}(0, \tau_r^{-1}).$$

- Parameters of interest are $\boldsymbol{\theta} = [\tau_q \quad \tau_r]^T$. For simplicity, we assume $p(\boldsymbol{\theta}) \propto 1$.
- $\mathbf{y}$ denotes the meas. sequence and $\mathbf{x}$ the state sequence.



- We get

$$p(\mathbf{x}|\boldsymbol{\theta}) = p(x_0)\prod_{k=1}^{T} p(\mathbf{x}_k|\mathbf{x}_{k-1}) = \mathcal{N}(x_0; \bar{x}_0, P_0)\prod_{k=1}^{T}\mathcal{N}(x_k; x_{k-1}, \tau_q^{-1})$$
$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}; \mathbf{x}, \tau_r^{-1}\mathbf{I})$$

Example – VB solution (2)

- An important part of VB is:

$$\begin{aligned}\log p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x}) &= \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) p(\mathbf{x}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) \\ &= \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) + \log p(\mathbf{x}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\end{aligned}$$

- Plugging in expressions from the previous slide yields:

$$\log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \log \mathcal{N}(\mathbf{y}; \mathbf{x}, \tau_r^{-1}\mathbf{I}) = \frac{T}{2}\log(\tau_r/(2\pi)) - \frac{\tau_r}{2}\sum_{k=1}^{T}(y_k - x_k)^2$$

$$\begin{aligned}\log p(\mathbf{x}|\boldsymbol{\theta}) &= \log \mathcal{N}(x_0; \bar{x}_0, P_0) + \sum_{k=1}^{T} \log \mathcal{N}(x_k; x_{k-1}, \tau_q^{-1}) \\ &= \log p(x_0) + \frac{T}{2}\log(\tau_q/(2\pi)) - \frac{\tau_q}{2}\sum_{k=1}^{T}(x_k - x_{k-1})^2\end{aligned}$$

- **Bottom line:** the logarithm turns $p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})$ into a simple sum!

Example – VB solution (3)

- According to the **VB algorithm**, we should set

$$q_x(\mathbf{x}) \propto \exp\left(\mathbb{E}_{q_\theta(\theta)}\left[\log\left[p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})\right]\right]\right).$$

- We can rewrite this as

$$\begin{aligned}
q_x(\mathbf{x}) \propto & p(x_0) \exp\left(-\mathbb{E}_{q_\theta(\boldsymbol{\theta})}\left[\frac{\tau_q}{2}\sum_{k=1}^{T}(x_k - x_{k-1})^2\right]\right) \\
& \exp\left(\mathbb{E}_{q_\theta(\boldsymbol{\theta})}\left[-\frac{\tau_r}{2}\sum_{k=1}^{T}(y_k - x_k)^2\right]\right) \\
\propto & p(x_0) \exp\left(-\frac{\mathbb{E}_{q_\theta(\boldsymbol{\theta})}[\tau_q]}{2}\sum_{k=1}^{T}(x_k - x_{k-1})^2\right) \\
& \exp\left(-\frac{\mathbb{E}_{q_\theta(\boldsymbol{\theta})}[\tau_r]}{2}\sum_{k=1}^{T}(y_k - x_k)^2\right)
\end{aligned}$$

- Do you recognize this as **something tractable**?

Example – VB solution (4)

- Let us introduce the notations $\bar{\tau}_q = \mathbb{E}_{q_\theta(\boldsymbol{\theta})}[\tau_q]$ and $\bar{\tau}_r = \mathbb{E}_{q_\theta(\boldsymbol{\theta})}[\tau_r]$.
- The previous equation can then be simplified to

$$q_x(\mathbf{x}) \propto p(x_0) \prod_{k=1}^{T} \mathcal{N}(x_k; x_{k-1}, \bar{\tau}_q^{-1}) \mathcal{N}(\mathbf{y}; \mathbf{x}, \bar{\tau}_r^{-1}\mathbf{I})$$

- **Conclusion:** to compute $q_x(\mathbf{x})$ we simply perform conventional (RTS) smoothing under the assumptions that $\tau_q = \bar{\tau}_q$ and $\tau_r = \bar{\tau}_r$.

Example – VB solution (5)

- According to the VB algorithm, we should set

$$q_\theta(\boldsymbol{\theta}) \propto \exp\left(\mathbb{E}_{q_x(\mathbf{x})}\left[\log\left[p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})\right]\right]\right).$$

- This simplifies to $q_\theta(\boldsymbol{\theta}) = q_{\tau_r}(\tau_r) q_{\tau_q}(\tau_q)$, where

$$\begin{aligned} q_{\tau_r}(\tau_r) &\propto \exp\left(\mathbb{E}_{q_x(\mathbf{x})}\left[\frac{T}{2}\log(\tau_r) - \frac{\tau_r}{2}\sum_{k=1}^{T}(y_k - x_k)^2\right]\right) \\ &\propto {\tau_r}^{T/2} \exp\left(-\frac{\tau_r}{2}\mathbb{E}_{q_x(\mathbf{x})}\left[\sum_{k=1}^{T}(y_k - x_k)^2\right]\right) \\ &\propto \text{Gam}\left(\tau_r; \frac{T+2}{2}, \frac{1}{2}\mathbb{E}_{q_x(\mathbf{x})}\left[\sum_{k=1}^{T}(y_k - x_k)^2\right]\right) \end{aligned}$$

Example – VB solution (6)

- According to the **VB algorithm**, we should set

$$q_\theta(\boldsymbol{\theta}) \propto \exp\left(\mathbb{E}_{q_x(\mathbf{x})}\left[\log\left[p(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})\right]\right]\right).$$

- Using the above derivations, we can show that

$$q_\theta(\boldsymbol{\theta}) = \text{Gam}\left(\tau_r; \frac{T+2}{2}, \frac{1}{2}b_r\right) \text{Gam}\left(\tau_q; \frac{T+2}{2}, \frac{1}{2}b_q\right)$$

where

$$b_r = \mathbb{E}_{q_x(\mathbf{x})}\left[\sum_{k=1}^{T}(y_k - x_k)^2\right] \quad \text{and} \quad b_q = \mathbb{E}_{q_x(\mathbf{x})}\left[\sum_{k=1}^{T}(x_k - x_{k-1})^2\right]$$

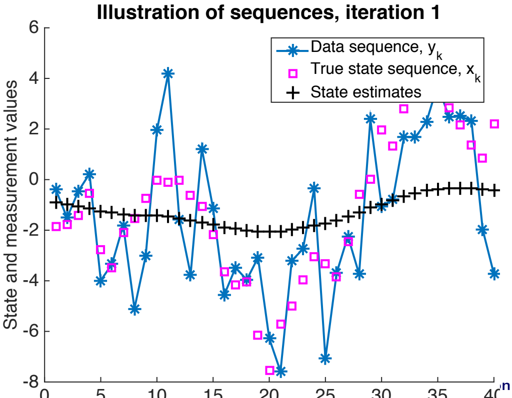
- It follows that $\mathbb{E}_{q_\theta}[\boldsymbol{\theta}] = (T+2)\begin{bmatrix} b_r^{-1} \\ b_q^{-1} \end{bmatrix}$. Is this reasonable?

Example – illustration of VB solution

- We can now study how the algorithm performs on an example.
- The true precisions were $\boldsymbol{\theta} = \begin{bmatrix}\tau_r \\ \tau_q\end{bmatrix} = \begin{bmatrix}1/4 \\ 1\end{bmatrix}$ and we initiated the algorithm with $\boldsymbol{\theta} = \begin{bmatrix}1/10 \\ 10\end{bmatrix}$. That is, with very little motion noise.

- **Our θ estimates:**

Iter. 1: $\bar{\boldsymbol{\theta}} = \begin{bmatrix}0.14 & 2.07\end{bmatrix}^T$
Iter. 2: $\bar{\boldsymbol{\theta}} = \begin{bmatrix}0.18 & 1.32\end{bmatrix}^T$
Iter. 3: $\bar{\boldsymbol{\theta}} = \begin{bmatrix}0.21 & 1.24\end{bmatrix}^T$
Iter. 4: $\bar{\boldsymbol{\theta}} = \begin{bmatrix}0.23 & 1.26\end{bmatrix}^T$
Iter. 5: $\bar{\boldsymbol{\theta}} = \begin{bmatrix}0.23 & 1.30\end{bmatrix}^T$
⋮
Iter. 10: $\bar{\boldsymbol{\theta}} = \begin{bmatrix}0.23 & 1.34\end{bmatrix}^T$



Final remarks on VB

Pros:

- VB is simple to employ in a wide range of contexts and often yields efficient algorithms.
- It is decreases the KL-divergence in every iteration and is thus guaranteed to converge (to a certain KLD).
- There is an alternative perspective on VB, as a maximizer of a lower bound on $p(\mathbf{y})$. That bound is useful for model selection.

Cons:

- It is a relatively crude approximation for at least two reasons: 1) the assumed factorisation breaks existing dependencies 2) it minimises the "wrong" KLD.
- For many problems, it is sensitive to initialization and may get stuck in a local minima.

Learning objectives

After this lecture you should be able to

- formulate a factor graph (FG) given a factorization of a function,
- explain why it is important to make use of the structure/sparseness of a problem,
- describe how the sum-product algorithm works on a factor graph (without loops),
- summarize the basic ideas behind variational Bayes.

A selection of references

General introductions

- Koller, Daphne, and Nir Friedman. Probabilistic graphical models: principles and techniques. MIT press, 2009.
- Bishop, C. M. Pattern recognition and machine learning. Springer, 2006.

Factor graphs and BP:

- Kschischang, Frank R., Brendan J. Frey, and H-A. Loeliger. "Factor graphs and the sum-product algorithm." IEEE Transactions on information theory 47.2 (2001): 498-519.
- Yedidia, Jonathan S., William T. Freeman, and Yair Weiss. "Understanding belief propagation and its generalizations." Exploring artificial intelligence in the new millennium 8 (2003): 236-239.

Basic principles and ideas, VB:

- Beal, M. J. Variational algorithms for approximate Bayesian inference. University of London, 2003.
- Winn, J. M., and Bishop, Christopher M.. "Variational message passing." Journal of Machine Learning Research. 2005.

Deep learning:

- Kingma, Diederik P., and Max Welling. "Auto-encoding Variational Bayes." arXiv preprint arXiv:1312.6114 (2013).