Ch. 19: Approximate Inference

Slides about
I. Goodfellow, Y. Bengio, *Deep Learning*, Ch. 19

by J. Cid-Sueiro
Universidad Carlos III de Madrid

jesus.cid@uc3m.es

May 21, 2018

Contents

Inference

- Deep learning (DL) models have visible ($\mathbf{v}$) and latent variables ($\mathbf{h}$).
- **Inference**: the difficult problem of computing $p(\mathbf{h}|\mathbf{v})$ or related expectations.
- **Intractable inference** problems in DL arise from complex interactions between latent variables in a structured graphical model.

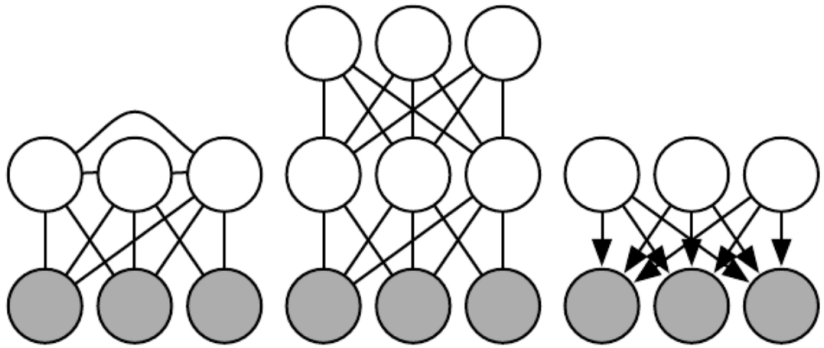


Figure 19.1: Intractable inference problems in deep learning are usually the result of interactions between latent variables in a structured graphical model. These interactions can be due to edges directly connecting one latent variable to another or longer paths that are activated when the child of a V-structure is observed. *(Left)*A **semi-restricted Boltzmann machine** (Osindero and Hinton, 2008) with connections between hidden units. These direct connections between latent variables make the posterior distribution intractable because of large cliques of latent variables. *(Center)*A deep Boltzmann machine, organized into layers of variables without intralayer connections, still has an intractable posterior distribution because of the connections between layers. *(Right)*This directed model has interactions between latent variables when the visible variables are observed, because every two latent variables are coparents. Some probabilistic models are able

Evidence Lower Bound (ELBO)

- **Data log-likelihood**:
  - Assume we would like to compute the log-probability of the observed data, $\log p(\mathbf{v}; \boldsymbol{\theta})$.
  - Problem: marginalize out $\mathbf{h}$ is costly.
  - Solution: compute a lower bound $L(\mathbf{v}, \boldsymbol{\theta}, q)$ on $\log p(\mathbf{v}; \boldsymbol{\theta})$.

Evidence Lower Bound (ELBO)

- **Data log-likelihood**:
  - Assume we would like to compute the log-probability of the observed data, $\log p(\mathbf{v}; \boldsymbol{\theta})$.
  - Problem: marginalize out $\mathbf{h}$ is costly.
  - Solution: compute a lower bound $L(\mathbf{v}, \boldsymbol{\theta}, q)$ on $\log p(\mathbf{v}; \boldsymbol{\theta})$.
- **Evidence Lower Bound** (ELBO)
  - (a.k.a. negative variational free energy)
  - It is defined as

$$L(\mathbf{v}, \boldsymbol{\theta}, q) = \log p(\mathbf{v}; \boldsymbol{\theta}) - \mathrm{DKL}(q(\mathbf{h}|\mathbf{v}) \| p(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta}))$$

    where q is an arbitrary probability distribution over $\mathbf{h}$.
  - Surprisingly, L can be considerably easier to compute for some distributions q.

Inference as optimization

$$L(\mathbf{v}, \boldsymbol{\theta}, q) = \log p(\mathbf{v}; \boldsymbol{\theta}) - \mathrm{DKL}(q(\mathbf{h}|\mathbf{v}) \| p(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta}))$$

- For any q, L is a lower bound on the likelihood.
- For $q = p$, the approximation is perfect.

$$p(\mathbf{h}|\mathbf{v}) = \arg\max_q L(\mathbf{v}, \boldsymbol{\theta}, q)$$

$$\log p(\mathbf{v}; \boldsymbol{\theta}) = \max_q L(\mathbf{v}, \boldsymbol{\theta}, q)$$

$\Rightarrow$ Inference is stated as an optimization problem.

Alternative form

- L is tractable, for some choices of q, using the following alternative form:

$$
\begin{aligned}
L(\mathbf{v}, \boldsymbol{\theta}, q) &= \log p(\mathbf{v}; \boldsymbol{\theta}) - \mathrm{DKL}(q(\mathbf{h}|\mathbf{v}) \| p(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta})) \\
&= \log p(\mathbf{v}; \boldsymbol{\theta}) - \mathbb{E}_{\mathbf{h} \sim q} \left[\log \frac{q(\mathbf{h}|\mathbf{v})}{p(\mathbf{h}|\mathbf{v})} \right] \\
&= \log p(\mathbf{v}; \boldsymbol{\theta}) - \mathbb{E}_{\mathbf{h} \sim q} \left[\log \frac{q(\mathbf{h}|\mathbf{v}) p(\mathbf{v}; \theta)}{p(\mathbf{h}, \mathbf{v}; \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_{\mathbf{h} \sim q} [\log p(\mathbf{h}, \mathbf{v}; \boldsymbol{\theta})] - \mathbb{E}_{\mathbf{h} \sim q} [\log q(\mathbf{h}|v)] \\
&= \mathbb{E}_{\mathbf{h} \sim q} [\log p(\mathbf{h}, \mathbf{v})] + H(q)
\end{aligned}
$$

Contents

Expectation-Maximization (EM)

- **EM**: An algorithm to compute the maximum likelihood (ML) estimate

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{v}; \boldsymbol{\theta})$$

  - It is a **coordinate ascent** algorithm, that alternates between two steps until convergence:

Expectation-Maximization (EM)

- **EM**: An algorithm to compute the maximum likelihood (ML) estimate

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{v}; \boldsymbol{\theta})$$

  - It is a **coordinate ascent** algorithm, that alternates between two steps until convergence:
- Algorithm:
  - Initialization: $\boldsymbol{\theta}^{(0)}$.
  - **E-step** (expectation): take

$$q^{(k)} = \arg\max_{q} L(\mathbf{v}, \boldsymbol{\theta}^{(k)}, q) = p(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta}^{(k)})$$

Expectation-Maximization (EM)

- **EM**: An algorithm to compute the maximum likelihood (ML) estimate

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{v}; \boldsymbol{\theta})$$

  - It is a **coordinate ascent** algorithm, that alternates between two steps until convergence:
- Algorithm:
  - Initialization: $\boldsymbol{\theta}^{(0)}$.
  - **E-step** (expectation): take

$$q^{(k)} = \arg\max_{q} L(\mathbf{v}, \boldsymbol{\theta}^{(k)}, q) = p(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta}^{(k)})$$

  - **M-step** (maximization): take

$$\boldsymbol{\theta}^{(k)} = \arg\max_{\boldsymbol{\theta}} L(\mathbf{v}, \boldsymbol{\theta}, q^{(k)})$$

- Not an approximate inference algorithm, but also based on ELBO maximization.

Expectation-Maximization

- **Variants**:
  - **Full EM**: analytical maximization. Only for some model families.
  - **Incremental EM**: Replaces maximization by an incremental progress.
  - **Stochastic gradient ascent**: can be seen as a special case of EM where the M-step takes a single (stochastic) gradient step.
- Insights.
  - Model parameters are updated to improve the likelihood of a *completed* dataset, where all missing variables have their values provided by an estimate of the posterior distribution.
  - We can continue to use one value of q even after we have moved to a different value of θ.
  - Large M-step updates are used in classical machine learning, though rarely in DL, because most models are too complex to admit a tractable solution for an optimal large M-step update.

Contents

MAP inference

- **Idea**: Instead of computing $p(\mathbf{h}|\mathbf{v})$, compute the single most likely value of the latent variables: $\mathbf{h}^* = arg\,\max_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})$.
- **Variational interpretation**:

MAP inference

- **Idea**: Instead of computing $p(\mathbf{h}|\mathbf{v})$, compute the single most likely value of the latent variables: $\mathbf{h}^* = arg\,\max_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})$.
- **Variational interpretation**:
  - Restrict q to be a Dirac distribution: $q(\mathbf{h}|\mathbf{v}) = \delta(\mathbf{h} - \boldsymbol{\mu})$
  - **Coordinate ascent**:
    - Optimize q

$$q^{(k)} = \arg\max_q L(\mathbf{v}, \boldsymbol{\theta}^{(k)}, q)$$
$$= \arg\max_q \{\mathbb{E}_{\mathbf{h}\sim q}[\log p(\mathbf{h}, \mathbf{v})] + H(q)\}$$

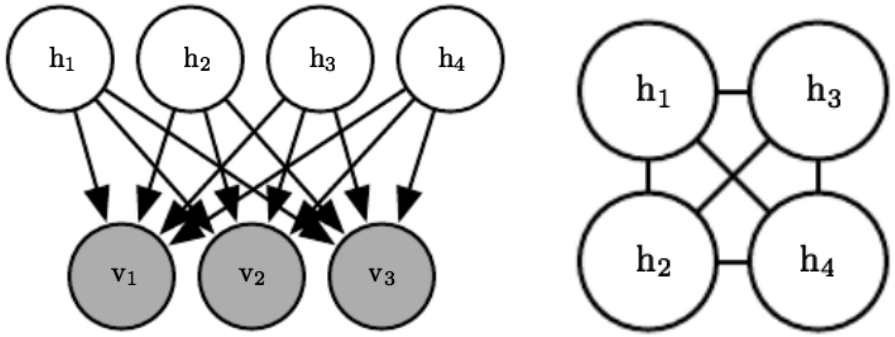
      which is equivalent (?) to

$$\boldsymbol{\mu}^* = \arg\max_{\boldsymbol{\mu}} \log p(\mathbf{h} = \boldsymbol{\mu}, \mathbf{v})$$

      which is a MAP inference problem.
    - Optimize $\boldsymbol{\theta}^{(k)} = \arg\max_{\boldsymbol{\theta}} L(\mathbf{v}, \boldsymbol{\theta}, q^{(k)})$.
  - Note that, though the computation of $\mathbf{h}^*$ may be exact, inference is approximate:

MAP inference in Deep learning

- MAP inference is primarily used in DL for sparse coding.
- **Sparse coding model**:
  - A linear factor model that imposes a sparsity-inducing prior on its hidden units. E.g.:
    - Factorial Laplace prior: $p(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}$
    - Linear gaussian observation: $p(\mathbf{v}|\mathbf{h}) = N(\mathbf{v}; \mathbf{W}\mathbf{h} + \mathbf{b}, \beta^{-1}\mathbf{I})$.



  - Contrary to the Gaussian case, $p(\mathbf{h}|\mathbf{v})$ is intractable.
    - Given $\mathbf{v}$, all the hidden units form a massive clique.
    - Since $p(\mathbf{h}|\mathbf{v})$ is intractable, we cannot use exact ML learning.
    - Instead, we use MAP inference.

MAP inference in Sparse Coding

- Optimization:
  - Concatenating vectors $\mathbf{h}$ into matrix $\mathbf{H}$, and vectors $\mathbf{v}$ into $\mathbf{V}$, the sparse coding learning process consists of minimizing

$$J(\mathbf{H}, \mathbf{W}) = \sum_{ij} |H_{ij}| + \alpha \sum_{ij} (V_{ij} - (\mathbf{H}\mathbf{W}^\intercal)_{ij} - b_i)^2$$

  - Minimization of J with respect to $\mathbf{H}$ and $\mathbf{W}$ is, in general, not a convex problem.
- **Coordinate Descent**:
  - Minimize wrt $\mathbf{W}$ (linear regression).
  - Minimize wrt $\mathbf{H}$ (requires specialized algorithms such as the feature-sign search algorithm (Lee et al., 2007)).

Contents

Variational Learning

- Core idea:
  - Maximize L over a restricted family of distributions q.
  - Choose the family so that it is easy to compute $\mathbb{E}_q \log p(\mathbf{h}, \mathbf{v})$.

Variational Learning

- Core idea:
  - Maximize L over a restricted family of distributions q.
  - Choose the family so that it is easy to compute $\mathbb{E}_q \log p(\mathbf{h}, \mathbf{v})$.
- Approaches:
  - **Mean field**: q is a factorial distribution:

$$q(\mathbf{h}|\mathbf{v}) = \prod_i q(h_i|\mathbf{v})$$

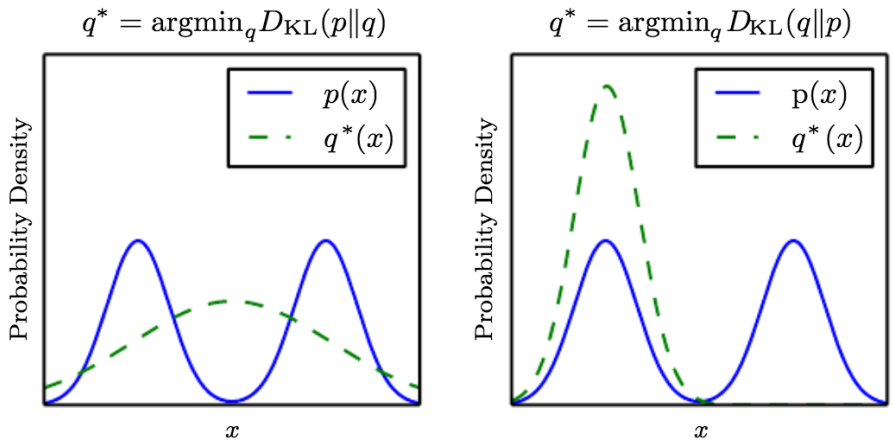
  - **Structured Variational inference** (Saul and Jordan, 1996): Impose a more general graphical model structure on q.
- A parametric form of q is not required.
  - **Discrete** latent variables: optimization over a finite number of variables describing q.
  - **Continuous** latent variables: calculus of variations to perform optimization over a space of functions.

Variational Learning

- Because $L(\mathbf{v}, \boldsymbol{\theta}, q) = \log p(\mathbf{v}; \boldsymbol{\theta}) - DKL(q(\mathbf{h}|\mathbf{v}) \| p(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta}))$, maximizing L wrt q is equivalent to minimizing $DKL(q(\mathbf{h}|\mathbf{v}) \| p(\mathbf{h}|\mathbf{v}))$.
  - Variational learning fits q to p in the opposite direction of the DKL used in ML to fit a model to data ($DKL(p_{\text{data}} \| p_{\text{model}})$).
    - ML: high probability everywhere that the data has high probability
    - Variational inference: low probability everywhere the true posterior has low probability.

Variational Learning

- Because $L(\mathbf{v}, \boldsymbol{\theta}, q) = \log p(\mathbf{v}; \boldsymbol{\theta}) - DKL(q(\mathbf{h}|\mathbf{v}) \| p(\mathbf{h}|\mathbf{v}; \boldsymbol{\theta}))$, maximizing L wrt q is equivalent to minimizing $DKL(q(\mathbf{h}|\mathbf{v}) \| p(\mathbf{h}|\mathbf{v}))$.
  - Variational learning fits q to p in the opposite direction of the DKL used in ML to fit a model to data ($DKL(p_{\text{data}} \| p_{\text{model}})$).
    - ML: high probability everywhere that the data has high probability
    - Variational inference: low probability everywhere the true posterior has low probability.



- The choice is motivated by computational reasons: we cannot guarantee a reduced-cost approach to computing $DKL(p|q)$.

Discrete Latent Variables

- General procedure:
  - Define a distribution q (e.g., a lookup table over discrete states).
  - Optimize parameters with a standard optimization algorithm (e.g. gradient descent).
  - Because this optimization must occur in the inner loop of a learning algorithm, it must be very fast.

Discrete Latent Variables

- General procedure:
  - Define a distribution q (e.g., a lookup table over discrete states).
  - Optimize parameters with a standard optimization algorithm (e.g. gradient descent).
  - Because this optimization must occur in the inner loop of a learning algorithm, it must be very fast.
- Simplest case:
  - $\mathbf{h}$ binary
  - q factorizes over each individual h_i.
  - Parameter vector $\hat{\mathbf{h}}$ of probability values $\hat{h}_i = q(h_i = 1|\mathbf{v})$.

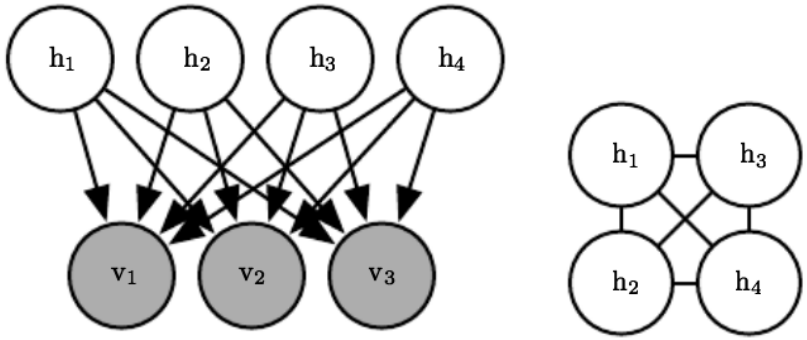
Binary Sparse Coding

- **Binary Sparse Coding**:
  - Gaussian Linear model with $\mathbf{h} \in \{0, 1\}^m$ and

$$p(h_i = 1) = \sigma(b_i)$$

$$p(\mathbf{v}|\mathbf{h}) = \mathcal{N}(\mathbf{v}; \mathbf{W}\mathbf{h}, \beta^{-1})$$

where $\mathbf{b}$, $\mathbf{W}$ and β are learnable parameters.

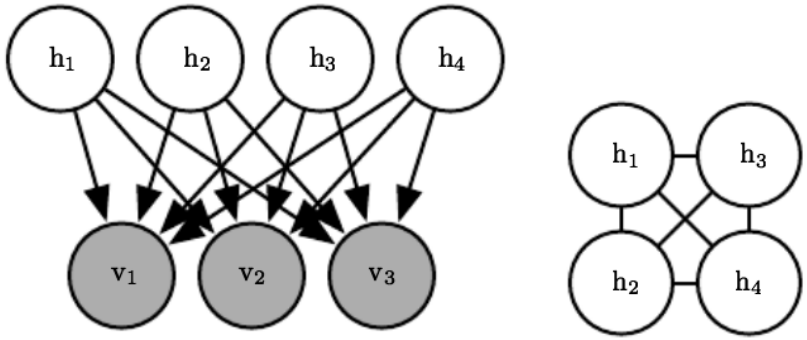


Binary Sparse Coding

- **Binary Sparse Coding**:
  - Gaussian Linear model with $\mathbf{h} \in \{0, 1\}^m$ and

$$p(h_i = 1) = \sigma(b_i)$$
$$p(\mathbf{v}|\mathbf{h}) = \mathcal{N}(\mathbf{v}; \mathbf{W}\mathbf{h}, \beta^{-1})$$

where $\mathbf{b}$, $\mathbf{W}$ and β are learnable parameters.



It can be shown (see book) that

$$\frac{\partial}{\partial b_i} \log p(\mathbf{v}) = \mathbb{E}_{h \sim p(\mathbf{h}|\mathbf{v})} \frac{\partial}{\partial b_i} \log p(\mathbf{h}) \tag{1}$$

which requires computing expectations wrt $p(\mathbf{h}|\mathbf{v})$: intractable.

Approximate Inference

- **Mean field approximation**:
  - $q(\mathbf{h}|\mathbf{v}) = \prod_i q(h_i|\mathbf{v})$
  - $q(h_i|\mathbf{v})$: Bernoulli with parameter $\hat{h}_i$ or $z_i = \sigma^{-1}(\hat{h}_i)$.

Approximate Inference

- **Mean field approximation**:
  - $q(\mathbf{h}|\mathbf{v}) = \prod_i q(h_i|\mathbf{v})$
  - $q(h_i|\mathbf{v})$: Bernoulli with parameter $\hat{h}_i$ or $z_i = \sigma^{-1}(\hat{h}_i)$.

$$\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q) \tag{19.29}$$

$$= \mathbb{E}_{\mathbf{h}\sim q}[\log p(\boldsymbol{h}, \boldsymbol{v})] + H(q) \tag{19.30}$$

$$= \mathbb{E}_{\mathbf{h}\sim q}[\log p(\boldsymbol{h}) + \log p(\boldsymbol{v} \mid \boldsymbol{h}) - \log q(\boldsymbol{h} \mid \boldsymbol{v})] \tag{19.31}$$

$$= \mathbb{E}_{\mathbf{h}\sim q}\left[\sum_{i=1}^{m} \log p(h_i) + \sum_{i=1}^{n} \log p(v_i \mid \boldsymbol{h}) - \sum_{i=1}^{m} \log q(h_i \mid \boldsymbol{v})\right] \tag{19.32}$$

$$= \sum_{i=1}^{m} \left[\hat{h}_i(\log \sigma(b_i) - \log \hat{h}_i) + (1 - \hat{h}_i)(\log \sigma(-b_i) - \log(1 - \hat{h}_i))\right] \tag{19.33}$$

$$+ \mathbb{E}_{\mathbf{h}\sim q}\left[\sum_{i=1}^{n} \log \sqrt{\frac{\beta_i}{2\pi}} \exp\left(-\frac{\beta_i}{2}(v_i - \boldsymbol{W}_{i,:}\boldsymbol{h})^2\right)\right] \tag{19.34}$$

$$= \sum_{i=1}^{m} \left[\hat{h}_i(\log \sigma(b_i) - \log \hat{h}_i) + (1 - \hat{h}_i)(\log \sigma(-b_i) - \log(1 - \hat{h}_i))\right] \tag{19.35}$$

$$+ \frac{1}{2}\sum_{i=1}^{n}\left[\log \frac{\beta_i}{2\pi} - \beta_i\left(v_i^2 - 2v_i \boldsymbol{W}_{i,:}\hat{\boldsymbol{h}} + \sum_j \left[W_{i,j}^2 \hat{h}_j + \sum_{k\neq j} W_{i,j}W_{i,k}\hat{h}_j\hat{h}_k\right]\right)\right]. \tag{19.36}$$

  - ELBO is tractable $\rightarrow$ we can run gradient ascent on both $\mathbf{v}$ and $\hat{\mathbf{h}}$.

Approximate Inference (II)

- **Fixed-point iterations**:
  - Usually, however, we do not do this, for two reasons:
    - **Scalability**: It requires storing $\hat{\mathbf{h}}$ for each $\mathbf{v}$. It is dificult to scale learning algorithms to billions of examples if we must remember a dynamically updated vector associated with each example.
    - **Speed**: In some applications, we may need to compute $\hat{\mathbf{h}}$ in real time.

Approximate Inference (II)

- **Fixed-point iterations**:
  - Usually, however, we do not do this, for two reasons:
    - **Scalability**: It requires storing $\hat{\mathbf{h}}$ for each $\mathbf{v}$. It is dificult to scale learning algorithms to billions of examples if we must remember a dynamically updated vector associated with each example.
    - **Speed**: In some applications, we may need to compute $\hat{\mathbf{h}}$ in real time.
  - Alternative: estimate with fixed-point equations.
    - **Goal**: find a local maximum as a solution of $\nabla_{\mathbf{h}} L(\mathbf{v}, \boldsymbol{\theta}, \hat{\mathbf{h}}) = 0$
    - **Solution**: No explicit solution for $\hat{\mathbf{h}}$, but we can solve for $\hat{h}_i$

$$\frac{\partial}{\partial \hat{h}_i} L(\mathbf{v}, \boldsymbol{\theta}, \hat{\mathbf{h}}) = 0$$

which leads to

$$\hat{h}_i = \sigma\left(b_i + \mathbf{v}^\intercal \boldsymbol{\beta} \mathbf{W}_{:,i} - \frac{1}{2}\mathbf{W}_{:,i}^\intercal \boldsymbol{\beta} \mathbf{W}_{:,i} - \sum_{j \neq i} \mathbf{W}_{:,j}^\intercal \boldsymbol{\beta} \mathbf{W}_{:,i} \hat{h}_j\right)$$

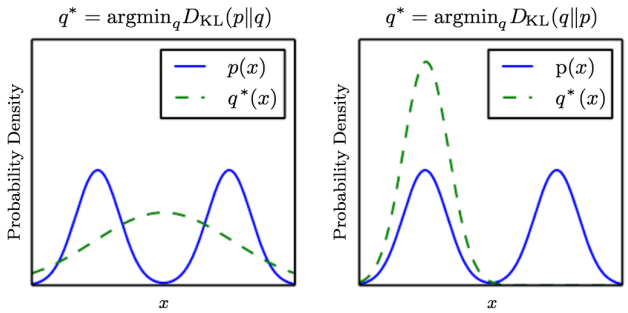
and iterate.

Approximate Inference (III)

- Connecting RNNs and inference in graphical models.
  - The mean field fixed-point equations define a RNN for inference.
  - The RNN for binary sparse coding repeatedly updates the hidden units based on the values of the neighboring hidden units.
  - The input always sends a fixed message of $\mathbf{v}^\top \beta \mathbf{W}$. to the hidden units, and these constantly update the message they send to each other.

Approximate Inference (III)

- Connecting RNNs and inference in graphical models.
  - The mean field fixed-point equations define a RNN for inference.
  - The RNN for binary sparse coding repeatedly updates the hidden units based on the values of the neighboring hidden units.
  - The input always sends a fixed message of $\mathbf{v}^\top \boldsymbol{\beta} \mathbf{W}$. to the hidden units, and these constantly update the message they send to each other.
  - If $\mathbf{W}_{:,i} = \mathbf{W}_{:,j}$, units $\hat{h}_i$ and $\hat{h}_j$ inhibit each other, competing to explain the input: only the unit that explains the input best will remain active.
    - This competition tries to capture explaining away interactions in the binary sparse coding posterior, that should cause a multimodal posterior
    - Unfortunately, explaining away interactions cannot be modeled by the factorial q, which is forced to choose one mode to model.



Calculus of Variations

- **Functions and functionals**
  - Many ML techniques are based on minimizing a function $J(\boldsymbol{\theta})$ with respect to input vector $\boldsymbol{\theta} \in \mathbb{R}^n$.
  - In some cases, we actually want to solve for a function $f(\mathbf{x})$ (e.g. when optimizing for q).
  - A function of a function f is known as a functional $J[f]$.

Calculus of Variations

- **Functions and functionals**
  - Many ML techniques are based on minimizing a function $J(\boldsymbol{\theta})$ with respect to input vector $\boldsymbol{\theta} \in \mathbb{R}^n$.
  - In some cases, we actually want to solve for a function $f(\mathbf{x})$ (e.g. when optimizing for q).
  - A function of a function f is known as a functional $J[f]$.
  - Functional (a.k.a variational) derivatives, of a functional $J[f]$ can be taken with respect to individual values of the function $f(\mathbf{x})$ at any $\mathbf{x}$.

$$\frac{\partial}{\partial f(\mathbf{x})} J$$

  - For differentiable functions $f(\mathbf{x})$ and $g(y, \mathbf{x})$ with continuous derivatives,

$$\frac{\partial}{\partial f(\mathbf{x})} \int g(f(\mathbf{x}'), \mathbf{x}') d\mathbf{x}' = \frac{\partial}{\partial y} g(f(\mathbf{x}), \mathbf{x})$$

Functional optimization

- We can optimize a functional by solving for the function where the functional derivative at every point is equal to zero.
- Example:
  - Finding the PDF over $x \in \mathbb{R}$ with maximal differential entropy $H[p] = \mathbb{E}_x \log p(x)$, with two constraints:
    - Fixed mean μ (for unicity)
    - Fixed variance, σ (to avoid degeneracy)

Functional optimization

- We can optimize a functional by solving for the function where the functional derivative at every point is equal to zero.
- Example:
  - Finding the PDF over $x \in \mathbb{R}$ with maximal differential entropy $H[p] = \mathbb{E}_x \log p(x)$, with two constraints:
    - Fixed mean μ (for unicity)
    - Fixed variance, σ (to avoid degeneracy)
  - Because of the constraints over p, we need to use Lagrange multipliers.

$$\begin{aligned} L[p] =& H[p] + \lambda_1 \left(\int p(x) dx - 1 \right) + \lambda_2 (\mathbb{E}[x] - \mu) \\ & + \lambda_3 \left(\mathbb{E}[(x-\mu)^2] - \sigma^2 \right) \\ =& \int \left(\lambda_1 p(x) + \lambda_2 p(x) x + \lambda_3 p(x)(x-\mu)^2 - p(x) \log p(x) \right) dx \\ & - \lambda_1 - \mu\lambda_2 - \sigma^2 \lambda_3 \end{aligned}$$

Calculus of variations

- Example (cont.):
  - By differentiation, we get:

$$p(x) = \exp\left(\lambda_1 + \lambda_2 x + \lambda_3\left((x-\mu)^2 - 1\right)\right)$$

that is, $x \sim \mathcal{N}(x; \mu, \sigma^2)$.

Continuous latent variables

- For inference over graphical models with continuous latent variables, we need to maximize L with respect to $q(\mathbf{h}|\mathbf{v})$.

Continuous latent variables

- For inference over graphical models with continuous latent variables, we need to maximize L with respect to $q(\mathbf{h}|\mathbf{v})$.
- **Mean-field fixed-point updates**:
  - Making the mean field approximation

$$q(\mathbf{h}|\mathbf{v}) = \prod_i q(h_i|\mathbf{v})$$

    and fixing $q(h_j|\mathbf{v})$ for all $j \neq i$, then the optimal $q(h_i|\mathbf{v})$ may be obtained by normalizing the unnormalized distribution

$$\tilde{q}(h_i|\mathbf{v}) = \exp\left(\mathbb{E}_{h_{-i} \sim q(h_{-i}|\mathbf{v})} \log \tilde{p}(\mathbf{v}, \mathbf{h})\right)$$

    which is valid for any probabilistic model.
  - This fixed-point equation, also tells us the functional form of the optimal solution, regardless of the fixed-point derivation.
  - We could take the functional form from that equation but regard some of the values that appear in it as parameters, and apply any optimization algorithm.

Derivation

- ELBO:

$$
\begin{aligned}
L(\mathbf{v}, q, \boldsymbol{\theta}) &= \mathbb{E}_{\mathbf{h}\sim q}[\log p(\mathbf{h}, \mathbf{v})] + H(q) \\
&= \int \prod_i q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h} - \int \prod_i q(h_i|\mathbf{v}) \log \prod_i q(h_i|\mathbf{v}) d\mathbf{h} \\
&= \int \prod_i q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h} - \sum_i \int q(h_i|\mathbf{v}) \log q(h_i|\mathbf{v}) d\mathbf{h}
\end{aligned}
$$

Derivation

- ELBO:

$$L(\mathbf{v}, q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{h}\sim q}[\log p(\mathbf{h}, \mathbf{v})] + H(q)$$
$$= \int \prod_i q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h} - \int \prod_i q(h_i|\mathbf{v}) \log \prod_i q(h_i|\mathbf{v}) d\mathbf{h}$$
$$= \int \prod_i q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h} - \sum_i \int q(h_i|\mathbf{v}) \log q(h_i|\mathbf{v}) d\mathbf{h}$$

- By functional differentiation of the Lagrangian, we get

$$\frac{\partial}{\partial q}\left(L + \lambda\left(\int q dh_j - 1\right)\right) = \int \prod_{i\neq j} q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h}_{-i} - \frac{\partial}{\partial q} q \log q\Big|_{q=q(h_i|\mathbf{v})} + \lambda$$
$$= \int \prod_{i\neq j} q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h}_{-i} - \log q(h_i|\mathbf{v}) - 1 + \lambda =$$
$$= \mathbb{E}_{h_{-1}\sim q(h_{-i}|\mathbf{v})} \log p(\mathbf{h}, \mathbf{v}) - \log q(h_i|\mathbf{v}) - 1 + \lambda = 0$$

Derivation

- ELBO:

$$L(\mathbf{v}, q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{h}\sim q}[\log p(\mathbf{h}, \mathbf{v})] + H(q)$$
$$= \int \prod_i q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h} - \int \prod_i q(h_i|\mathbf{v}) \log \prod_i q(h_i|\mathbf{v}) d\mathbf{h}$$
$$= \int \prod_i q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h} - \sum_i \int q(h_i|\mathbf{v}) \log q(h_i|\mathbf{v}) d\mathbf{h}$$

- By functional differentiation of the Lagrangian, we get

$$\frac{\partial}{\partial q}\left(L + \lambda\left(\int q dh_j - 1\right)\right) = \int \prod_{i\neq j} q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h}_{-i} - \frac{\partial}{\partial q} q \log q\Big|_{q=q(h_i|\mathbf{v})} + \lambda$$
$$= \int \prod_{i\neq j} q(h_i|\mathbf{v}) \log p(\mathbf{h}, \mathbf{v}) d\mathbf{h}_{-i} - \log q(h_i|\mathbf{v}) - 1 + \lambda =$$
$$= \mathbb{E}_{h_{-1}\sim q(h_{-i}|\mathbf{v})} \log p(\mathbf{h}, \mathbf{v}) - \log q(h_i|\mathbf{v}) - 1 + \lambda = 0$$

and, thus,

$$q(h_i|\mathbf{v}) = \exp\left(\mathbb{E}_{h_{-1}\sim q(h_{-i}|\mathbf{v})} \log p(\mathbf{h}, \mathbf{v}) - 1 + \lambda\right)$$

Continuous latent variables

- Example (a tractable case used for illustration):
  - $p(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mathbf{0}, \mathbf{I})$, $\mathbf{h} = (h_1, h_2)$
  - $p(v|\mathbf{h}) = N(v; \mathbf{w}^\intercal \mathbf{h}; 1)$
  - **True posterior**:

Continuous latent variables

- Example (a tractable case used for illustration):
  - $p(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mathbf{0}, \mathbf{I})$, $\mathbf{h} = (h_1, h_2)$
  - $p(v|\mathbf{h}) = N(v; \mathbf{w}^\intercal \mathbf{h}; 1)$
  - **True posterior**:

$$p(\mathbf{h}|v) \; \alpha \; p(\mathbf{h}, v) = p(h_1)p(h_2)p(v|\mathbf{h})$$
$$\alpha \; \exp\left(\frac{1}{2}\left[h_1^2 + h_2^2 + (v - h_1 w_1 - h_2 w_2)^2\right]\right)$$

    which does not factorize over $\mathbf{h}$.
  - **Mean field approximation**:

$$\tilde{q}(h_1|v) = \exp\left(\mathbb{E}_{h_2 \sim q(h_2|v)} \log \tilde{p}(v, \mathbf{h})\right)$$
$$= \exp\left(-\frac{1}{2}\mathbb{E}_{h_2 \sim q(h_2|v)}\left[h_1^2 + h_2^2 + (v - h_1 w_1 - h_2 w_2)^2\right]\right)$$

    which is a Gaussian $\rightarrow$ we can take $q(\mathbf{h}|v) = \mathcal{N}(\mathbf{h}; \mu, \beta^{-1})$ where μ and diagonal β are variational parameters, and optimize.
  - Recall: we did not ever assume that q would be Gaussian.

Interactions between learning and inference

- Using approximate inference affects the accuracy of the inference algorithm.
  - The training algorithm tends to adapt the model in a way that makes the approximating assumptions become more true.
    - When training the parameters, variational learning increases $\mathbb{E}_{\mathbf{h}\sim q} \log p(\mathbf{v}, \mathbf{h})$
    - For a specific $\mathbf{v}$, this increases $p(\mathbf{h}|\mathbf{v})$ for values of $\mathbf{h}$ that have high probability under $q(\mathbf{h}|\mathbf{v})$ and decreases $p(\mathbf{h}|\mathbf{v})$ for values of $\mathbf{h}$ that have low probability under $q(\mathbf{h}|\mathbf{v})$.
    - This behavior causes our approximating assumptions to become self-fulfilling prophecies.
    - If we train the model with a unimodal approximate posterior, we will obtain a model with a true posterior that is far closer to unimodal than we would have obtained by training the model with exact inference.
  - Computing the true amount of harm imposed on a model by a variational approximation is thus very difficult.

Interactions between learning and inference

- Summarizing:
  - The variational approximation is accurate for the specific value of $\boldsymbol{\theta}$ that we obtained from the learning process, but
  - ... this does not imply that it is accurate in general or that it did little harm to the learning process.
- To measure the true amount of harm induced by the variational approximation, we would need to know $\boldsymbol{\theta}^* = \max_{\boldsymbol{\theta}} \log p(\mathbf{v}; \boldsymbol{\theta})$.
  - It is possible for $L(\mathbf{v}, \boldsymbol{\theta}, q) \approx \log p(\mathbf{v}; \boldsymbol{\theta})$ and $\log p(\mathbf{v}; \boldsymbol{\theta}) \ll \log p(\mathbf{v}; \boldsymbol{\theta}^*)$ to hold simultaneously.
  - If $\max_q L(\mathbf{v}, \boldsymbol{\theta}^*, q) \ll \log p(\mathbf{v}; \boldsymbol{\theta}^*)$, because $\boldsymbol{\theta}^*$ induces too complicated of a posterior distribution for our q family to capture, then the learning process will never approach $\boldsymbol{\theta}^*$.
  - Such a problem is very difficult to detect, because we can only know for sure that it happened if we have a superior learning algorithm that can find $\boldsymbol{\theta}^*$ for comparison

Contents

Learned Approximate Inference

- Fixed-point equations or gradient-based optimization can be very expensive and time consuming.
- Alternative: learning to perform approximate inference.
  - Idea: we can think of the optimization process as a function f that maps an input $\mathbf{v}$ to an approximate distribution $q^* = \arg\max_q L(\mathbf{v}, q)$. $\Rightarrow$ approximate it with a neural network that implements an approximation $\hat{f}(\mathbf{v}; \boldsymbol{\theta})$.

Wake-sleep

- One of the main difficulties with training a model to infer **h** from **v** is that we do not have a supervised training set with observations of **h**.
- The mapping from **v** to **h** depends on the choice of model family, and evolves throughout the learning process as θ changes.
- **Wake-sleep algorithm** (Hinton et al., 1995b; Frey et al., 1996):
  - Draw samples of both **h** and **v** from the model distribution.
  - Train the inference network to perform the reverse mapping: predicting which **h** caused the present **v**.
  - Drawback: we train the inference network on values of **v** that have high probability under the model. Early in learning, the model distribution will not resemble the data distribution, so the inference network will not have an opportunity to learn on samples that resemble data.