02477 – Bayesian Machine Learning: Lecture 12

Michael Riis Andersen

Technical University of Denmark,
DTU Compute, Department of Applied Math and Computer Science

Variational inference in modern machine learning

- *Variational inference* (VI) has been the topic for the last two weeks
- Variational inference has many applications in modern machine learning
  1. Variational autoencoders (VAEs)
  2. Diffusion models
  3. Bayesian neural networks
  4. ...
- Common challenges
  1. Highly non-linear models
  2. Large-scale models
  3. Hugh datasets
- Last bits of VI theory today: *Black-box variational inference* (BBVI)

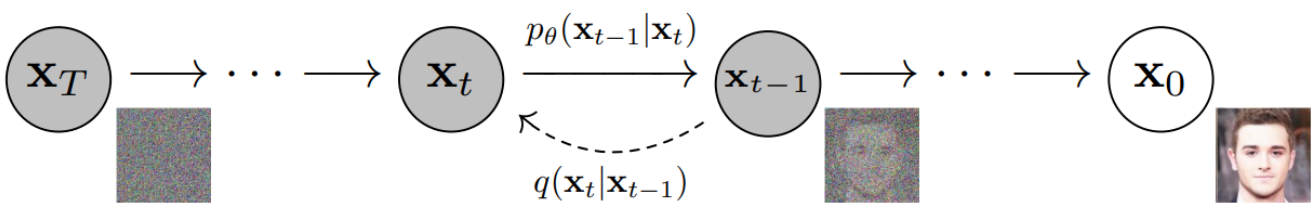


Figure 2: The directed graphical model considered in this work.

From Jonathan Ho et al (2020)

Outline

1. Variational inference and divergences
   - Properties of the KL-divergence
   - KL-divergences in other contexts (detour)

2. Black-box variational inference
   - The evidence lower bound and entropy
   - Dealing with expectations
   - Stochastic gradients
   - A few words on stochastic optimization
   - Mini-batching for large scale inference

Variational inference and divergences

Variational inference and divergences: Properties of the KL-divergence

Measuring distance between probability distributions

- How do we measure "distance" between probability distributions $\mathbb{D}[q, p]$?

Measuring distance between probability distributions

- How do we measure "distance" between probability distributions $\mathbb{D}[q, p]$?
- The choice of "distance" affects the properties of the approximation

Measuring distance between probability distributions

- How do we measure "distance" between probability distributions $\mathbb{D}[q, p]$?
- The choice of "distance" affects the properties of the approximation
- The *Kullback-Leibler* divergence (for continuous R.V.) is defined as

$$\mathrm{KL}[q||p] = \int q(\boldsymbol{z}) \log \left[\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})} \right] \mathrm{d}\boldsymbol{z}$$

Measuring distance between probability distributions

- How do we measure "distance" between probability distributions $\mathbb{D}[q, p]$?
- The choice of "distance" affects the properties of the approximation
- The *Kullback-Leibler* divergence (for continuous R.V.) is defined as

$$\mathrm{KL}[q||p] = \int q(\boldsymbol{z}) \log \left[\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})} \right] \mathrm{d}\boldsymbol{z}$$

- Properties
  1. Identity of indiscernibles

$$\mathrm{KL}[q||p] = 0 \qquad \Longleftrightarrow \qquad p = q \quad \text{(a.e)}$$

Measuring distance between probability distributions

- How do we measure "distance" between probability distributions $\mathbb{D}[q, p]$?
- The choice of "distance" affects the properties of the approximation
- The *Kullback-Leibler* divergence (for continuous R.V.) is defined as

$$\mathrm{KL}[q||p] = \int q(\boldsymbol{z}) \log \left[\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})} \right] \mathrm{d}\boldsymbol{z}$$

- Properties
  1. Identity of indiscernibles

$$\mathrm{KL}[q||p] = 0 \qquad \Longleftrightarrow \qquad p = q \quad \text{(a.e)}$$

  2. Non-negativity

$$\mathrm{KL}[q||p] \geq 0$$

Measuring distance between probability distributions

- How do we measure "distance" between probability distributions $\mathbb{D}[q, p]$?
- The choice of "distance" affects the properties of the approximation
- The *Kullback-Leibler* divergence (for continuous R.V.) is defined as

$$\mathrm{KL}[q||p] = \int q(\boldsymbol{z}) \log \left[\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})} \right] \mathrm{d}\boldsymbol{z}$$

- Properties
  1. Identity of indiscernibles

$$\mathrm{KL}[q||p] = 0 \qquad \Longleftrightarrow \qquad p = q \quad \text{(a.e)}$$

  2. Non-negativity

$$\mathrm{KL}[q||p] \geq 0$$

  3. Asymmetric

$$\mathrm{KL}[q||p] \neq \mathrm{KL}[p||q]$$

Properties of the KL divergence

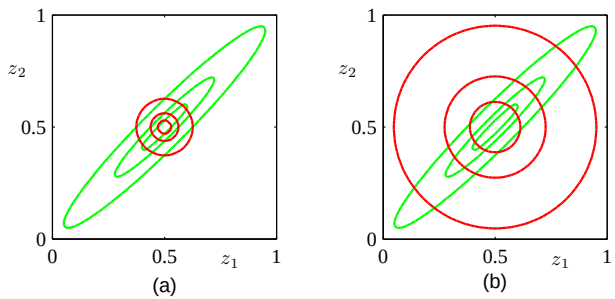
- The *variational approximation* q for target distribution $p \approx q$ is defined as

$$q_* = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}\left[q||p\right], \qquad \mathrm{KL}\left[q||p\right] = \int q(\boldsymbol{z}) \log\left[\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})}\right] \mathrm{d}\boldsymbol{z}$$

- Approximating a general Gaussian $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with a mean-field Gaussian

$$q(\boldsymbol{w}) = \mathcal{N}(w_1|m_1, v_1)\mathcal{N}(w_2|m_2, v_2)$$

- Which approximation q (in red) has the smallest $\mathrm{KL}\left[q||p\right]$ divergence? (a) or (b)?



The approximation q in red
The target p distribution in green

Properties of the KL divergence

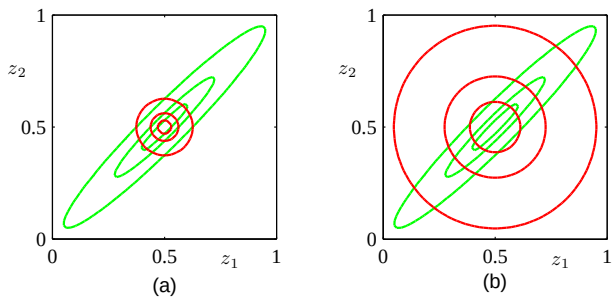
- The *variational approximation* q for target distribution $p \approx q$ is defined as

$$q_* = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}\left[q||p\right], \qquad \mathrm{KL}\left[q||p\right] = \int q(\boldsymbol{z}) \log\left[\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})}\right] \mathrm{d}\boldsymbol{z}$$

- Approximating a general Gaussian $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with a mean-field Gaussian

$$q(\boldsymbol{w}) = \mathcal{N}(w_1|m_1, v_1)\mathcal{N}(w_2|m_2, v_2)$$

- Which approximation q (in red) has the smallest $\mathrm{KL}\left[q||p\right]$ divergence? (a) or (b)?



The approximation q in red
The target p distribution in green

- (a) will be the minimizer of $\mathrm{KL}\left[q||p\right]$ (*mode-seeking behavour*)

Properties of the KL divergence

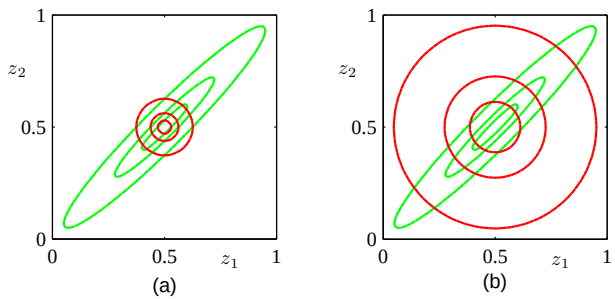
- The *variational approximation* q for target distribution $p \approx q$ is defined as

$$q_* = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}\left[q||p\right], \qquad \mathrm{KL}\left[q||p\right] = \int q(\boldsymbol{z}) \log\left[\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})}\right] \mathrm{d}\boldsymbol{z}$$

- Approximating a general Gaussian $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with a mean-field Gaussian

$$q(\boldsymbol{w}) = \mathcal{N}(w_1|m_1, v_1)\mathcal{N}(w_2|m_2, v_2)$$

- Which approximation q (in red) has the smallest $\mathrm{KL}\left[q||p\right]$ divergence? (a) or (b)?



The approximation q in red
The target p distribution in green

- (a) will be the minimizer of $\mathrm{KL}\left[q||p\right]$ (*mode-seeking behavour*)

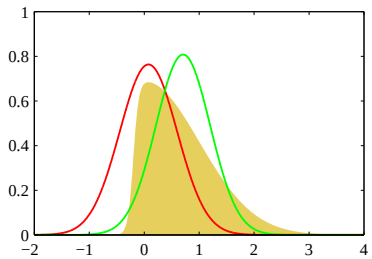
- (b) will be the minimizer of $\mathrm{KL}\left[p||q\right]$ (*mass-covering behavour*)

Common approximation properties

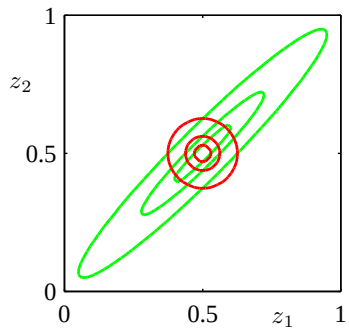
The *variational approximation* q for target distribution $p \approx q$ is defined as $q_* = \arg\min_{q \in \mathcal{Q}} \mathbb{D}\left[q||p\right]$

$$\mathrm{KL}\left[q||p\right] = \int q(\boldsymbol{z}) \log\left[\frac{q(\boldsymbol{z})}{p(\boldsymbol{z})}\right] \mathrm{d}\boldsymbol{z} \qquad \mathrm{KL}\left[p||q\right] = \int p(\boldsymbol{z}) \log\left[\frac{p(\boldsymbol{z})}{q(\boldsymbol{z})}\right] \mathrm{d}\boldsymbol{z}$$

1. The variational approximation (green) often covers to posterior mass of target (yellow) better than the Laplace approximation (red)

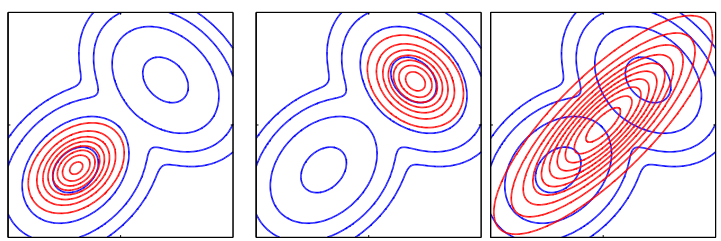


2. For the combination of *factorized variational families and KL* $[q||p]$, the variational approximation (red) often *underestimates* the variance of the target distribution (green)



(a)

3. Approximations (red) based on KL$[q||p]$ are often said to be *mode-seeking*, whereas KL$[p||q]$ are often said to be *mass-covering*



The α-divergence

- The Kullback-Leibler divergence is not the only possible choice for the divergence

$$\mathrm{KL}\left[q||p\right] = \int q(\boldsymbol{z}) \log \frac{q(\boldsymbol{z})}{p(\boldsymbol{z})} \mathrm{d}\boldsymbol{z}$$

- The *α-divergence* is defined as

$$\mathbb{D}_\alpha\left[p||q\right] = \frac{1}{\alpha(1-\alpha)} \int \alpha p(\boldsymbol{z}) + (1-\alpha) q(\boldsymbol{z}) - p(\boldsymbol{z})^\alpha q(\boldsymbol{z})^{1-\alpha} \mathrm{d}\boldsymbol{z}$$

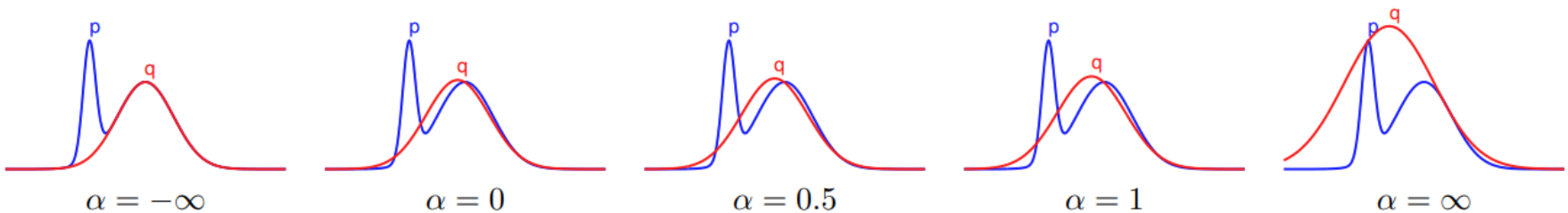
- Some interesting *special cases*

$$\lim_{\alpha \to 0} \mathbb{D}_\alpha\left[p||q\right] = \mathrm{KL}\left[q||p\right]$$

$$\lim_{\alpha \to 1} \mathbb{D}_\alpha\left[p||q\right] = \mathrm{KL}\left[p||q\right]$$

$$\mathbb{D}_{\frac{1}{2}}\left[p||q\right] = 2 \int \left(\sqrt{p(\boldsymbol{z})} - \sqrt{q(\boldsymbol{z})}\right) \mathrm{d}\boldsymbol{z} \qquad \text{(Hellinger distance)}$$

- α determines the properties of the resulting approximation $q^* = \arg\min_q \mathbb{D}_\alpha\left[p||q\right]$



T. Minka, 2005: Divergence measures and message passing

Variational inference and divergences: KL-divergences in other contexts (detour)

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\text{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$\text{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] = \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right]$$

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\mathrm{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$\begin{aligned}\mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\ &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p_D(\boldsymbol{x})\right] + \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right]\end{aligned}$$

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\mathrm{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$
\begin{aligned}
\mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p_D(\boldsymbol{x})\right] + \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] + K
\end{aligned}
$$

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\mathrm{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$
\begin{aligned}
\mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p_D(\boldsymbol{x})\right] + \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] + K \\
&= -\mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p(\mathbf{x}|\boldsymbol{\theta})\right] + K
\end{aligned}
$$

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\mathrm{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$
\begin{aligned}
\mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p_D(\boldsymbol{x})\right] + \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] + K \\
&= -\mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p(\mathbf{x}|\boldsymbol{\theta})\right] + K \\
&= -\frac{1}{N}\sum_{N=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta}) + K
\end{aligned}
$$

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\mathrm{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$
\begin{aligned}
\mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p_D(\boldsymbol{x})\right] + \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] + K \\
&= -\mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p(\mathbf{x}|\boldsymbol{\theta})\right] + K \\
&= -\frac{1}{N}\sum_{N=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta}) + K
\end{aligned}
$$

- Therefore,

$$
\hat{\theta}^* = \arg\min_{\theta} \mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right]
$$

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\mathrm{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$
\begin{aligned}
\mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p_D(\boldsymbol{x})\right] + \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] + K \\
&= -\mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p(\mathbf{x}|\boldsymbol{\theta})\right] + K \\
&= -\frac{1}{N}\sum_{N=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta}) + K
\end{aligned}
$$

- Therefore,

$$
\hat{\theta}^* = \arg\min_{\theta} \mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] \approx \arg\max_{\theta} \sum_{N=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta})
$$

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\mathrm{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$
\begin{aligned}
\mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p_D(\boldsymbol{x})\right] + \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] + K \\
&= -\mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p(\mathbf{x}|\boldsymbol{\theta})\right] + K \\
&= -\frac{1}{N}\sum_{N=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta}) + K
\end{aligned}
$$

- Therefore,

$$
\hat{\theta}^* = \arg\min_{\theta} \mathrm{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] \approx \arg\max_{\theta} \sum_{N=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta}) \equiv \hat{\theta}_{\mathrm{MLE}}
$$

De-tour: KL-divergences and maximum likelihood

- Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \sim p_{\text{D}}$. Assume we want to fit some parametric distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

$$\begin{aligned}
\text{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] &= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{p_D(\boldsymbol{x})}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p_D(\boldsymbol{x})\right] + \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{p_D(\boldsymbol{x})}\left[\log \frac{1}{p(\mathbf{x}|\boldsymbol{\theta})}\right] + K \\
&= -\mathbb{E}_{p_D(\boldsymbol{x})}\left[\log p(\mathbf{x}|\boldsymbol{\theta})\right] + K \\
&= -\frac{1}{N}\sum_{N=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta}) + K
\end{aligned}$$

- Therefore,

$$\hat{\theta}^* = \arg\min_{\theta} \text{KL}\left[p_D(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})\right] \approx \arg\max_{\theta} \sum_{N=1}^{N} \log p(\mathbf{x}_n|\boldsymbol{\theta}) \equiv \hat{\theta}_{\text{MLE}}$$

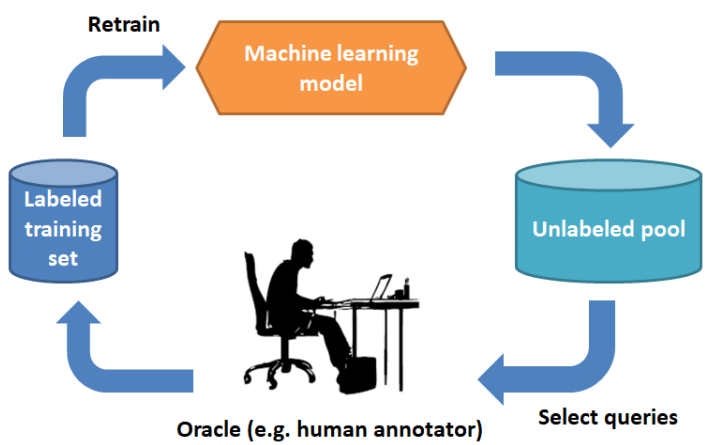
- Asymptotically ($N \to \infty$), *maximum likelihood learning* is equivalent to minimizing the *KL divergence between the true data distribution $p_D(\boldsymbol{x})$ and your model $p(\boldsymbol{x}|\boldsymbol{\theta})$*.

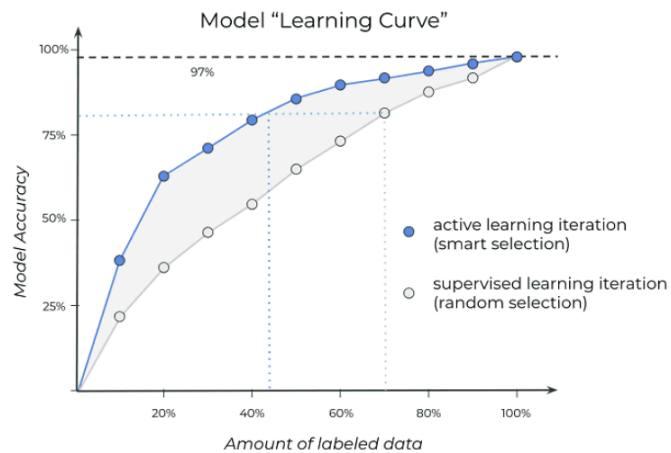
De-tour: Information gain

- Consider some Bayesian model with prior $p(\boldsymbol{w})$ and posterior $p(\boldsymbol{w}|\mathcal{D})$
- The KL divergence between the posterior and the prior is also sometimes called *information gain*

$$\mathrm{KL}\left[p(\boldsymbol{w}|\mathcal{D})||p(\boldsymbol{w})\right]$$

- ... and quantifies how much *information* we gain by moving from the prior to the posterior
- Often used in *active learning* and *optimal experiment design* to choose the most informative subset of data for more data-efficient learning.





Li et al, 2018: Reversed Active Learning based Atrous DenseNet for Pathological Image Classification

https://www.kdnuggets.com/2018/10/introduction-active-learning.html

Black-box variational inference

Free-form and fixed-form variational infernece

- In *free-form* VI, we use a *factorized approximation* for approximating the target distribution $p \equiv p(\boldsymbol{w}|\boldsymbol{y})$

$$q(\boldsymbol{w}) = \prod_{j=1}^{J} q(\boldsymbol{w}_j), \quad \text{where} \quad \boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_J]$$

- In *fixed-form* VI we choose a specific family distributions, e.g.

$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{V})$$

Free-form and fixed-form variational infernece

- In *free-form* VI, we use a *factorized approximation* for approximating the target distribution $p \equiv p(\boldsymbol{w}|\boldsymbol{y})$

$$q(\boldsymbol{w}) = \prod_{j=1}^{J} q(\boldsymbol{w}_j), \quad \text{where} \quad \boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_J]$$

- In *fixed-form* VI we choose a specific family distributions, e.g.

$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{V})$$

Free-form and fixed-form variational infernece

- In *free-form* VI, we use a *factorized approximation* for approximating the target distribution $p \equiv p(\boldsymbol{w}|\boldsymbol{y})$

$$q(\boldsymbol{w}) = \prod_{j=1}^{J} q(\boldsymbol{w}_j), \quad \text{where} \quad \boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_J]$$

- In *fixed-form* VI we choose a specific family distributions, e.g.

$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{V})$$

Free-form and fixed-form variational infernece

- In *free-form* VI, we use a *factorized approximation* for approximating the target distribution $p \equiv p(\boldsymbol{w}|\boldsymbol{y})$

$$q(\boldsymbol{w}) = \prod_{j=1}^{J} q(\boldsymbol{w}_j), \quad \text{where} \quad \boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_J]$$

- In *fixed-form* VI we choose a specific family distributions, e.g.

$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{V})$$

- ... and then evaluate and optimize the ELBO wrt. $q(\mathbf{w})$

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})]$$

Free-form and fixed-form variational infernece

- In *free-form* VI, we use a *factorized approximation* for approximating the target distribution $p \equiv p(\boldsymbol{w}|\boldsymbol{y})$

$$q(\boldsymbol{w}) = \prod_{j=1}^{J} q(\boldsymbol{w}_j), \quad \text{where} \quad \boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_J]$$

- In *fixed-form* VI we choose a specific family distributions, e.g.

$$q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{V})$$

- ... and then evaluate and optimize the ELBO wrt. $q(\mathbf{w})$

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})]$$

\+ Optimal functional form given assumptions (CAVI)

\+ Fast optimization

\- Requires model-specific derivations

\- Required integrals may be intractable

\- Optimal forms may not be "known" distributions

Black-box variational inference

- How can we avoid the need for model-specific derivations? How can we avoid the restrictions on the choice of models we can approximate?

Black-box variational inference

- How can we avoid the need for model-specific derivations? How can we avoid the restrictions on the choice of models we can approximate?

- Again, we use *fixed-form* families for continuous parameters, e.g. *mean-field* or *full-rank* Gaussians

$$q_{\text{MF}}(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i | m_i, v_i) \qquad q_{\text{FR}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} | \boldsymbol{m}, \boldsymbol{V})$$

Black-box variational inference

- How can we avoid the need for model-specific derivations? How can we avoid the restrictions on the choice of models we can approximate?

- Again, we use *fixed-form* families for continuous parameters, e.g. *mean-field* or *full-rank* Gaussians

$$q_{\text{MF}}(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i | m_i, v_i) \qquad\qquad q_{\text{FR}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} | \boldsymbol{m}, \boldsymbol{V})$$

- Optimize ELBO wrt. *variational parameters* $\boldsymbol{\lambda} = \{\boldsymbol{m}, \boldsymbol{V}\}$

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \mathcal{L}\left[q_{\boldsymbol{\lambda}}\right] = \arg\max_{\boldsymbol{\lambda}} \left\{\mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[\log p(\boldsymbol{y}, \boldsymbol{w})\right] - \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[\log q_{\boldsymbol{\lambda}}(\boldsymbol{w})\right]\right\}$$

Black-box variational inference

- How can we avoid the need for model-specific derivations? How can we avoid the restrictions on the choice of models we can approximate?

- Again, we use *fixed-form* families for continuous parameters, e.g. *mean-field* or *full-rank* Gaussians

$$q_{\text{MF}}(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i | m_i, v_i) \qquad\qquad q_{\text{FR}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} | \boldsymbol{m}, \boldsymbol{V})$$

- Optimize ELBO wrt. *variational parameters* $\boldsymbol{\lambda} = \{\boldsymbol{m}, \boldsymbol{V}\}$

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \mathcal{L}\left[q_{\boldsymbol{\lambda}}\right] = \arg\max_{\boldsymbol{\lambda}} \left\{\mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[\log p(\boldsymbol{y}, \boldsymbol{w})\right] - \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[\log q_{\boldsymbol{\lambda}}(\boldsymbol{w})\right]\right\}$$

- We want to find an "automatic" method for

  1. evaluating $\mathcal{L}\left[q_{\boldsymbol{\lambda}}\right]$
  2. computing gradients of $\mathcal{L}\left[q_{\boldsymbol{\lambda}}\right]$

Black-box variational inference

- How can we avoid the need for model-specific derivations? How can we avoid the restrictions on the choice of models we can approximate?

- Again, we use *fixed-form* families for continuous parameters, e.g. *mean-field* or *full-rank* Gaussians

$$q_{\text{MF}}(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i | m_i, v_i) \qquad\qquad q_{\text{FR}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} | \boldsymbol{m}, \boldsymbol{V})$$

- Optimize ELBO wrt. *variational parameters* $\boldsymbol{\lambda} = \{\boldsymbol{m}, \boldsymbol{V}\}$

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \mathcal{L}\left[q_{\boldsymbol{\lambda}}\right] = \arg\max_{\boldsymbol{\lambda}} \left\{\mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[\log p(\boldsymbol{y}, \boldsymbol{w})\right] - \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[\log q_{\boldsymbol{\lambda}}(\boldsymbol{w})\right]\right\}$$

- We want to find an "automatic" method for
  1. evaluating $\mathcal{L}\left[q_{\boldsymbol{\lambda}}\right]$
  2. computing gradients of $\mathcal{L}\left[q_{\boldsymbol{\lambda}}\right]$

- ... allowing us to easily prototype, implement and test different models

Black-box variational inference: The evidence lower bound and entropy

The evidence lower bound and entropy

- The evidence lower bound $\mathcal{L}[q]$ is defined as

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})]$$

The evidence lower bound and entropy

- The evidence lower bound $\mathcal{L}[q]$ is defined as

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] + \mathcal{H}[q]$$

The evidence lower bound and entropy

- The evidence lower bound $\mathcal{L}[q]$ is defined as

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] + \mathcal{H}[q]$$

- Second term is called the (differential) *entropy* of q, i.e. $\mathcal{H}[q] \equiv -\mathbb{E}_q[\log q(\boldsymbol{w})]$

The evidence lower bound and entropy

- The evidence lower bound $\mathcal{L}[q]$ is defined as

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] + \mathcal{H}[q]$$

- Second term is called the (differential) *entropy* of q, i.e. $\mathcal{H}[q] \equiv -\mathbb{E}_q[\log q(\boldsymbol{w})]$

- For "named" distributions, like Gaussians, the entropy is often easy to calculate or can be looked up.

The evidence lower bound and entropy

- The evidence lower bound $\mathcal{L}[q]$ is defined as

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] + \mathcal{H}[q]$$

- Second term is called the (differential) *entropy* of q, i.e. $\mathcal{H}[q] \equiv -\mathbb{E}_q[\log q(\boldsymbol{w})]$

- For "named" distributions, like Gaussians, the entropy is often easy to calculate or can be looked up.

- If $q(w_i) = \mathcal{N}(w_i|m_i, v_i)$, then

$$\mathcal{H}[q(w_i)] \equiv -\mathbb{E}_{q(w_i)}[\log q(w_i)] = \frac{1}{2}\log(2\pi e v_i)$$

The evidence lower bound and entropy

- The evidence lower bound $\mathcal{L}[q]$ is defined as

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] + \mathcal{H}[q]$$

- Second term is called the (differential) *entropy* of q, i.e. $\mathcal{H}[q] \equiv -\mathbb{E}_q[\log q(\boldsymbol{w})]$

- For "named" distributions, like Gaussians, the entropy is often easy to calculate or can be looked up.

- If $q(w_i) = \mathcal{N}(w_i|m_i, v_i)$, then

$$\mathcal{H}[q(w_i)] \equiv -\mathbb{E}_{q(w_i)}[\log q(w_i)] = \frac{1}{2}\log(2\pi e v_i)$$

- The entropy term and its gradient can often be computed analytically

The evidence lower bound and entropy

- The evidence lower bound $\mathcal{L}[q]$ is defined as

$$\mathcal{L}[q] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] - \mathbb{E}_q[\log q(\boldsymbol{w})] = \mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] + \mathcal{H}[q]$$

- Second term is called the (differential) *entropy* of q, i.e. $\mathcal{H}[q] \equiv -\mathbb{E}_q[\log q(\boldsymbol{w})]$

- For "named" distributions, like Gaussians, the entropy is often easy to calculate or can be looked up.

- If $q(w_i) = \mathcal{N}(w_i|m_i, v_i)$, then

$$\mathcal{H}[q(w_i)] \equiv -\mathbb{E}_{q(w_i)}[\log q(w_i)] = \frac{1}{2}\log(2\pi e v_i)$$

- The entropy term and its gradient can often be computed analytically

- Therefore, we will focus on dealing with the expected log joint term: $\mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})]$

Quiz

Quiz time!

Spend 4 minutes answering the following questions in the quiz

DTU Learn - Quiz - Lecture12: Variational families

Black-box variational inference: Dealing with expectations

How to deal with the expected log joint

- We can approximate the first term using Monte Carlo samples:

$$\mathbb{E}_q \left[\log p(\boldsymbol{y}, \boldsymbol{w})\right] \approx \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}),$$

where

$$\boldsymbol{w}^{(s)} \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \quad \text{for} \quad s = 1, \ldots, S$$

How to deal with the expected log joint

- We can approximate the first term using Monte Carlo samples:

$$\mathbb{E}_q \left[\log p(\boldsymbol{y}, \boldsymbol{w})\right] \approx \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}),$$

  where

$$\boldsymbol{w}^{(s)} \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \quad \text{for} \quad s = 1, \ldots, S$$

- Example: if $q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{V})$, we sample $\boldsymbol{w}^{(s)} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{V})$ and plug them into the sum above

How to deal with the expected log joint

- We can approximate the first term using Monte Carlo samples:

$$\mathbb{E}_q \left[\log p(\boldsymbol{y}, \boldsymbol{w})\right] \approx \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}),$$

  where

$$\boldsymbol{w}^{(s)} \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \quad \text{for} \quad s = 1, \ldots, S$$

- Example: if $q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{V})$, we sample $\boldsymbol{w}^{(s)} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{V})$ and plug them into the sum above

- Easy to implement and fits the bill

  1. All we need is to be able to evaluate to the log joint $p(\boldsymbol{y}, \boldsymbol{w})$
  2. Not restricted by existence of closed-form analytical results for expectations
  3. No model-specific derivations

How to deal with the expected log joint

- We can approximate the first term using Monte Carlo samples:

$$\mathbb{E}_q \left[\log p(\boldsymbol{y}, \boldsymbol{w})\right] \approx \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}),$$

  where

$$\boldsymbol{w}^{(s)} \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \quad \text{for} \quad s = 1, \ldots, S$$

- Example: if $q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{V})$, we sample $\boldsymbol{w}^{(s)} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{V})$ and plug them into the sum above

- Easy to implement and fits the bill
  1. All we need is to be able to evaluate to the log joint $p(\boldsymbol{y}, \boldsymbol{w})$
  2. Not restricted by existence of closed-form analytical results for expectations
  3. No model-specific derivations

- How about the gradients?

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}} \left[\log p(\boldsymbol{y}, \boldsymbol{w})\right]$$

Black-box variational inference: Stochastic gradients

How to compute the gradients?

- First, we can use *Leibniz'* rule to change order of the gradient and the integral

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] \equiv \nabla_{\boldsymbol{\lambda}} \int q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w}$$

How to compute the gradients?

- First, we can use *Leibniz'* rule to change order of the gradient and the integral

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \nabla_{\boldsymbol{\lambda}} \int q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\ &= \int \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w}\end{aligned}$$

How to compute the gradients?

- First, we can use *Leibniz'* rule to change order of the gradient and the integral

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \nabla_{\boldsymbol{\lambda}} \int q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\ &= \int \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w}\end{aligned}$$

- We can't estimate the gradients directly using MC sampling. Why not?

How to compute the gradients?

- First, we can use *Leibniz'* rule to change order of the gradient and the integral

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \nabla_{\boldsymbol{\lambda}} \int q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\ &= \int \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w}\end{aligned}$$

- We can't estimate the gradients directly using MC sampling. Why not?

- In order to estimate a quantity using MC, we need to able to express it as an expectation. For example,

$$\mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] = \int q(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \approx \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}),$$

How to compute the gradients?

- First, we can use *Leibniz'* rule to change order of the gradient and the integral

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \nabla_{\boldsymbol{\lambda}} \int q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\ &= \int \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w}\end{aligned}$$

- We can't estimate the gradients directly using MC sampling. Why not?

- In order to estimate a quantity using MC, we need to able to express it as an expectation. For example,

$$\mathbb{E}_q[\log p(\boldsymbol{y}, \boldsymbol{w})] = \int q(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \approx \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}),$$

- To generate samples, we need a distribution, but $\nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ is not a distribution

The score function estimator

- *Log-derivative trick*: Using the chain-rule on the $\log q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ yields

$$\nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \frac{1}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

The score function estimator

- *Log-derivative trick*: Using the chain-rule on the $\log q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ yields

$$\nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \frac{1}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

- We can use this to re-write the gradient

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] \equiv \int \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w}$$

The score function estimator

- *Log-derivative trick*: Using the chain-rule on the $\log q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ yields

$$\nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \frac{1}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

- We can use this to re-write the gradient

$$\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \int \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int \frac{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w}
\end{aligned}$$

The score function estimator

- *Log-derivative trick*: Using the chain-rule on the $\log q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ yields

$$\nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \frac{1}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

- We can use this to re-write the gradient

$$\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \int \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int \frac{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \frac{1}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w}
\end{aligned}$$

The score function estimator

- *Log-derivative trick*: Using the chain-rule on the $\log q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ yields

$$\nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \frac{1}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

- We can use this to re-write the gradient

$$\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \int \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int \frac{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \frac{1}{q_{\boldsymbol{\lambda}}(\boldsymbol{w})} \nabla_{\boldsymbol{\lambda}} q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \quad \text{(log-derivative trick)}
\end{aligned}$$

The score function estimator

- *Log-derivative trick*: Using the chain-rule on the $\log q_{\lambda}(\boldsymbol{w})$ yields

$$\nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) = \frac{1}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w})$$

- We can use this to re-write the gradient

$$\begin{aligned}
\nabla_{\lambda} \mathbb{E}_{q_{\lambda}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \int \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int \frac{q_{\lambda}(\boldsymbol{w})}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\lambda}(\boldsymbol{w}) \frac{1}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\lambda}(\boldsymbol{w}) \nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \quad \text{(log-derivative trick)} \\
&= \mathbb{E}_{q_{\lambda}}[\nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w})] \quad \text{(Def. of expectation)}
\end{aligned}$$

The score function estimator

- *Log-derivative trick*: Using the chain-rule on the $\log q_{\lambda}(\boldsymbol{w})$ yields

$$\nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) = \frac{1}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w})$$

- We can use this to re-write the gradient

$$\begin{aligned}
\nabla_{\lambda} \mathbb{E}_{q_{\lambda}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \int \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int \frac{q_{\lambda}(\boldsymbol{w})}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\lambda}(\boldsymbol{w}) \frac{1}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\lambda}(\boldsymbol{w}) \nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \quad \text{(log-derivative trick)} \\
&= \mathbb{E}_{q_{\lambda}}[\nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w})] \quad \text{(Def. of expectation)} \\
&\approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}^s) \log p(\boldsymbol{y}, \boldsymbol{w}^s) \quad \text{where} \quad \boldsymbol{w}^s \sim q_{\lambda}
\end{aligned}$$

The score function estimator

- *Log-derivative trick*: Using the chain-rule on the $\log q_{\lambda}(\boldsymbol{w})$ yields

$$\nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) = \frac{1}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w})$$

- We can use this to re-write the gradient

$$\begin{aligned}
\nabla_{\lambda} \mathbb{E}_{q_{\lambda}}[\log p(\boldsymbol{y}, \boldsymbol{w})] &\equiv \int \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int \frac{q_{\lambda}(\boldsymbol{w})}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\lambda}(\boldsymbol{w}) \frac{1}{q_{\lambda}(\boldsymbol{w})} \nabla_{\lambda} q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \\
&= \int q_{\lambda}(\boldsymbol{w}) \nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w}) \mathrm{d}\boldsymbol{w} \quad \text{(log-derivative trick)} \\
&= \mathbb{E}_{q_{\lambda}}[\nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}) \log p(\boldsymbol{y}, \boldsymbol{w})] \quad \text{(Def. of expectation)} \\
&\approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\lambda} \log q_{\lambda}(\boldsymbol{w}^s) \log p(\boldsymbol{y}, \boldsymbol{w}^s) \quad \text{where} \quad \boldsymbol{w}^s \sim q_{\lambda}
\end{aligned}$$

- This is called the *score function gradient estimator*

The score function gradient estimator

- The score function gradient estimator is defined as

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}^s) \log p(\boldsymbol{y}, \boldsymbol{w}^s) \quad \text{where} \quad \boldsymbol{w}^s \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

The score function gradient estimator

- The score function gradient estimator is defined as

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}^s) \log p(\boldsymbol{y}, \boldsymbol{w}^s) \qquad \text{where} \qquad \boldsymbol{w}^s \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

- $\nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}$ can easily be calculated by hand or using automatic differentiation (e.g. Pytorch, Tensorflow, Autograd etc.)

The score function gradient estimator

- The score function gradient estimator is defined as

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}^s) \log p(\boldsymbol{y}, \boldsymbol{w}^s) \quad \text{where} \quad \boldsymbol{w}^s \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

- $\nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}$ can easily be calculated by hand or using automatic differentiation (e.g. Pytorch, Tensorflow, Autograd etc.)

- Only need to be able to evaluate $\log p(\boldsymbol{y}, \boldsymbol{w})$. No need for model-specific derivations

The score function gradient estimator

- The score function gradient estimator is defined as

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}^s) \ \log p(\boldsymbol{y}, \boldsymbol{w}^s) \qquad \text{where} \qquad \boldsymbol{w}^s \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w})$$

- $\nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}$ can easily be calculated by hand or using automatic differentiation (e.g. Pytorch, Tensorflow, Autograd etc.)
- Only need to be able to evaluate $\log p(\boldsymbol{y}, \boldsymbol{w})$. No need for model-specific derivations
- General properties
  1. Very general and unbiased
  2. Applies to both continuous and discrete parameters
  3. Often high variance
- Big picture: approximating posterior distributions of general models
  1. We have chosen a variational family
  2. We aim to maximize the ELBO using gradient-based methods
  3. Estimate gradients using the score function estimator without the need for model-specific derivations
  4. Drawback: stochastic gradients

The BBVI algorithm

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. score function estimator

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$

The BBVI algorithm

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. score function estimator

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)

The BBVI algorithm

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. score function estimator

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{w}^{(s)} \sim q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_t, \boldsymbol{v}_t)$ for $s = 1, \ldots, S$

The BBVI algorithm

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. score function estimator

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{w}^{(s)} \sim q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_t, \boldsymbol{v}_t)$ for $s = 1, \ldots, S$
  2. Estimate ELBO

$$\hat{\mathcal{L}}[q] = \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}) + \mathcal{H}[q]$$

The BBVI algorithm

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. score function estimator

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{w}^{(s)} \sim q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_t, \boldsymbol{v}_t)$ for $s = 1, \ldots, S$
  2. Estimate ELBO

$$\hat{\mathcal{L}}[q] = \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}) + \mathcal{H}[q]$$

  3. Estimate gradient

$$\nabla_{\boldsymbol{\lambda}} \hat{\mathcal{L}}[q] \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}^s) \; \log p(\boldsymbol{y}, \boldsymbol{w}^s) + \nabla_{\boldsymbol{\lambda}} \mathcal{H}[q]$$

The BBVI algorithm

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. score function estimator

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{w}^{(s)} \sim q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_t, \boldsymbol{v}_t)$ for $s = 1, \ldots, S$
  2. Estimate ELBO

$$\hat{\mathcal{L}}[q] = \frac{1}{S} \sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}) + \mathcal{H}[q]$$

  3. Estimate gradient

$$\nabla_{\boldsymbol{\lambda}} \hat{\mathcal{L}}[q] \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}^s) \ \log p(\boldsymbol{y}, \boldsymbol{w}^s) + \nabla_{\boldsymbol{\lambda}} \mathcal{H}[q]$$

  4. Update variational parameters to get $\boldsymbol{\lambda}_{t+1} = \{\boldsymbol{m}_{t+1}, \log \boldsymbol{v}_{t+1}\}$ using gradient estimate

The BBVI algorithm

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. score function estimator

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{w}^{(s)} \sim q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_t, \boldsymbol{v}_t)$ for $s = 1, \ldots, S$
  2. Estimate ELBO

$$\hat{\mathcal{L}}[q] = \frac{1}{S}\sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}) + \mathcal{H}[q]$$

  3. Estimate gradient

$$\nabla_{\boldsymbol{\lambda}} \hat{\mathcal{L}}[q] \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}^s)\ \log p(\boldsymbol{y}, \boldsymbol{w}^s) + \nabla_{\boldsymbol{\lambda}} \mathcal{H}[q]$$

  4. Update variational parameters to get $\boldsymbol{\lambda}_{t+1} = \{\boldsymbol{m}_{t+1}, \log \boldsymbol{v}_{t+1}\}$ using gradient estimate
- Predictions & model checking: Evaluate training error, validation errors etc

The BBVI algorithm

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. score function estimator

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{w}^{(s)} \sim q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_t, \boldsymbol{v}_t)$ for $s = 1, \ldots, S$
  2. Estimate ELBO

$$\hat{\mathcal{L}}[q] = \frac{1}{S}\sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}^{(s)}) + \mathcal{H}[q]$$

  3. Estimate gradient

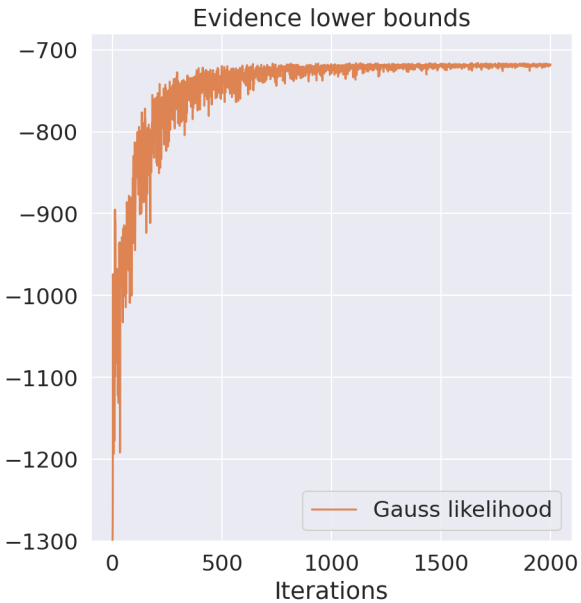
$$\nabla_{\boldsymbol{\lambda}} \hat{\mathcal{L}}[q] \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}^s)\ \log p(\boldsymbol{y}, \boldsymbol{w}^s) + \nabla_{\boldsymbol{\lambda}} \mathcal{H}[q]$$

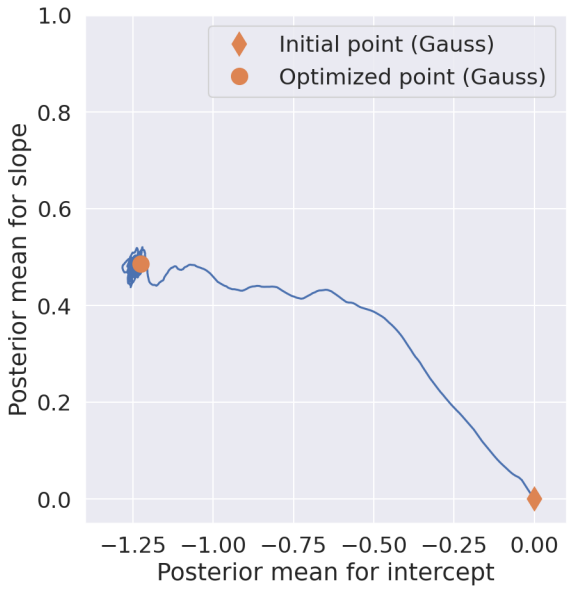
  4. Update variational parameters to get $\boldsymbol{\lambda}_{t+1} = \{\boldsymbol{m}_{t+1}, \log \boldsymbol{v}_{t+1}\}$ using gradient estimate
- Predictions & model checking: Evaluate training error, validation errors etc
- Go back and refine model if needed

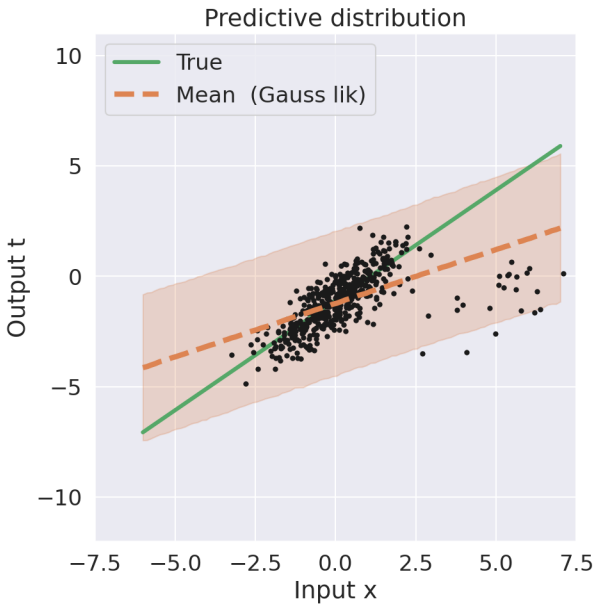
Example: Robust regression I

$$p(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} \mathcal{N}(y_n | \mathbf{w}^T \mathbf{x}_i, \sigma^2) \mathcal{N}(\boldsymbol{w} | \mathbf{0}, \kappa^2 \mathbf{I})$$

Example: Robust regression II



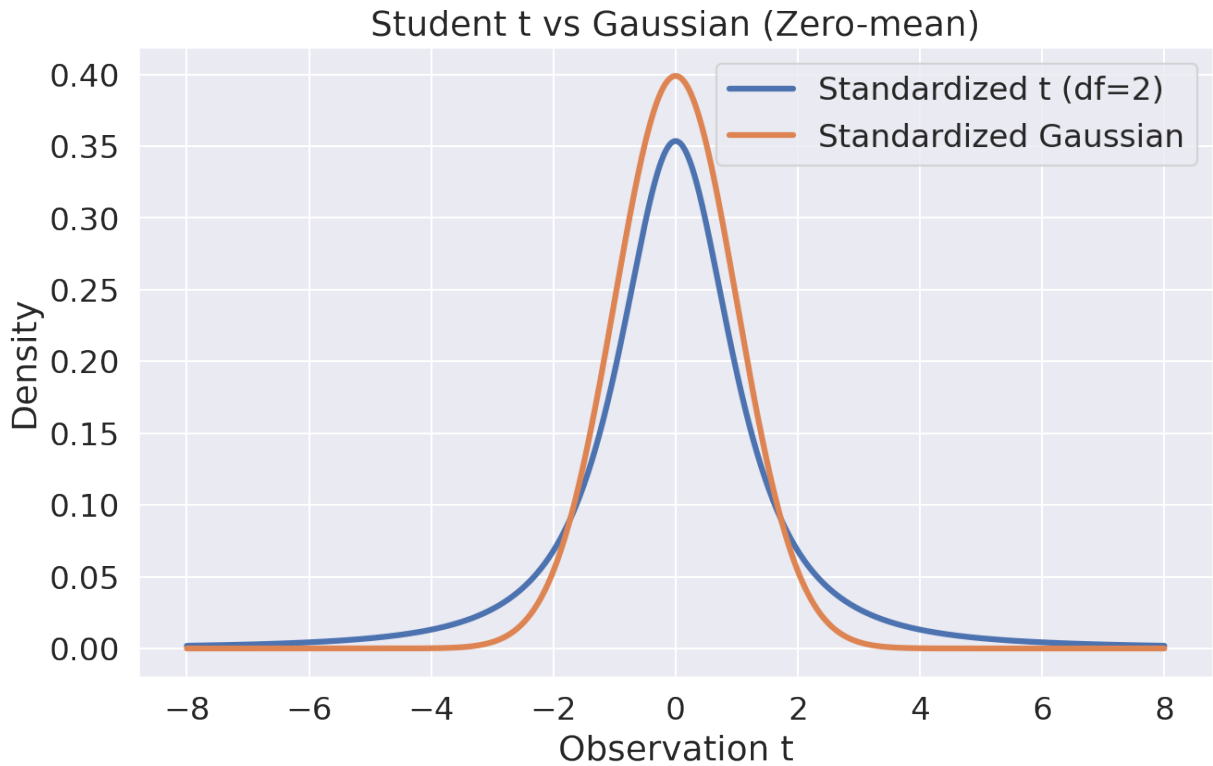




$$p(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} \mathcal{N}(y_n | \mathbf{w}^T \mathbf{x}_i, \sigma^2) \mathcal{N}(\boldsymbol{w} | \mathbf{0}, \kappa^2 \mathbf{I})$$

- Running BBVI with $S = 5$ samples for 2000 iterations

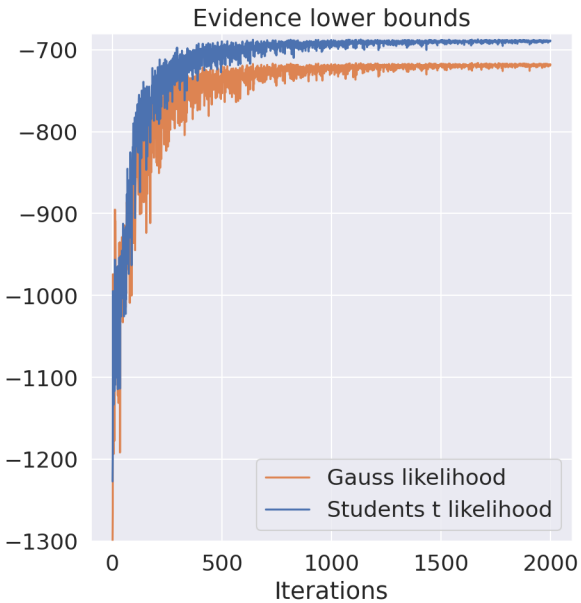
Example: Robust regression III

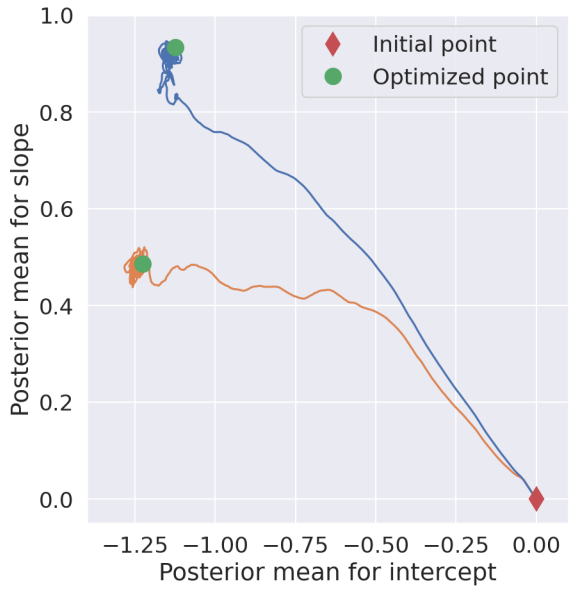


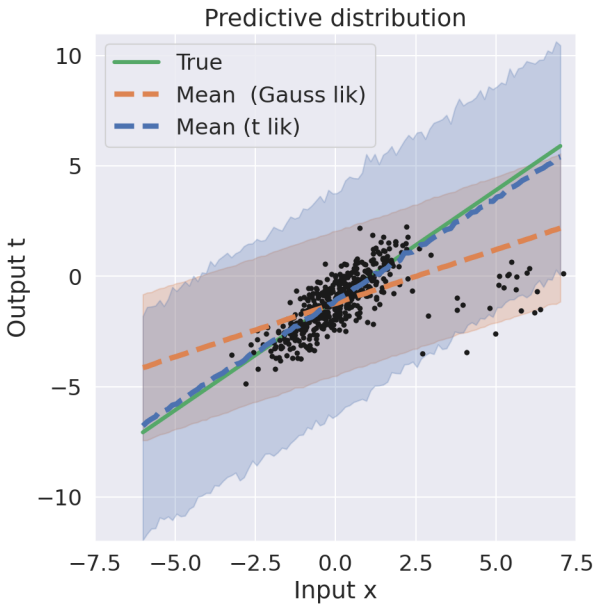
$$p_{\text{gauss}}(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} \mathcal{N}(y_n|\mathbf{w}^T\mathbf{x}_i, \sigma^2)\mathcal{N}(\boldsymbol{w}|\mathbf{0}, \kappa^2\mathbf{I})$$

$$p_{\text{student-t}}(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} t_2(y_n|\mathbf{w}^T\mathbf{x}_i, \sigma^2)\mathcal{N}(\boldsymbol{w}|\mathbf{0}, \kappa^2\mathbf{I})$$

Example: Robust regression IV







$$p_{\text{gauss}}(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} \mathcal{N}(y_n|\mathbf{w}^T\mathbf{x}_i, \sigma^2)\mathcal{N}(\boldsymbol{w}|\mathbf{0}, \kappa^2\mathbf{I})$$

$$p_{\text{student-t}}(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} t_2(y_n|\mathbf{w}^T\mathbf{x}_i, \sigma^2)\mathcal{N}(\boldsymbol{w}|\mathbf{0}, \kappa^2\mathbf{I})$$

- Running BBVI with $S = 5$ samples for 2000 iterations

Text classification: Spam detection

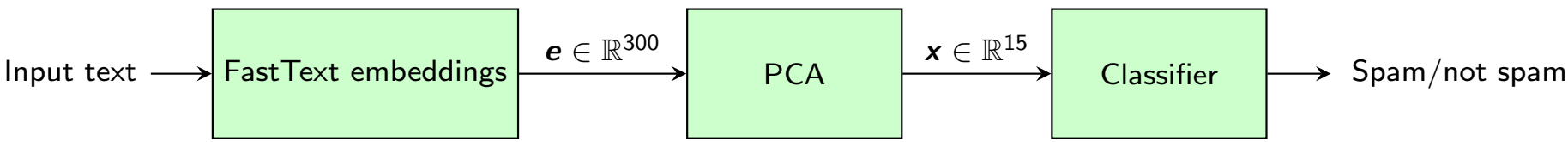
- In the exercise notebook, we will build a simple spam detector based on an SMS dataset

- Text examples

  1. "urgent! your mobile number has been awarded with a £2000 prize guaranteed. call 09061790121 from land line. claim 3030. valid 12hrs only 150ppm"
  2. "do we have any spare power supplies"
  3. "merry christmas to u too annie!"

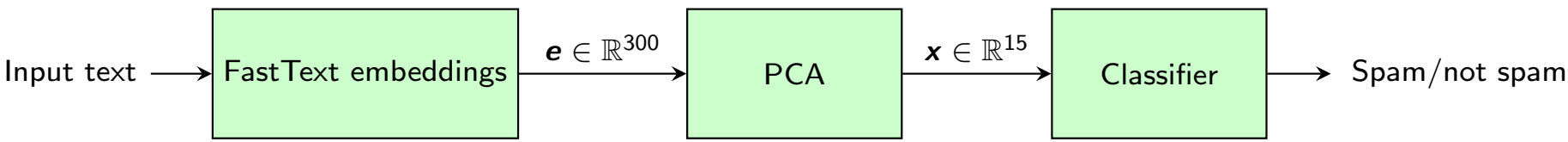
Text classification: Spam detection

- In the exercise notebook, we will build a simple spam detector based on an SMS dataset
- Text examples
  1. "urgent! your mobile number has been awarded with a £2000 prize guaranteed. call 09061790121 from land line. claim 3030. valid 12hrs only 150ppm"
  2. "do we have any spare power supplies"
  3. "merry christmas to u too annie!"
- Transfer learning: Embed texts using FastText embedding (www.fasttext.cc)



Text classification: Spam detection

- In the exercise notebook, we will build a simple spam detector based on an SMS dataset
- Text examples
  1. "urgent! your mobile number has been awarded with a £2000 prize guaranteed. call 09061790121 from land line. claim 3030. valid 12hrs only 150ppm"
  2. "do we have any spare power supplies"
  3. "merry christmas to u too annie!"
- Transfer learning: Embed texts using FastText embedding (www.fasttext.cc)



- Which likelihood to use to gain robustness to outliers?

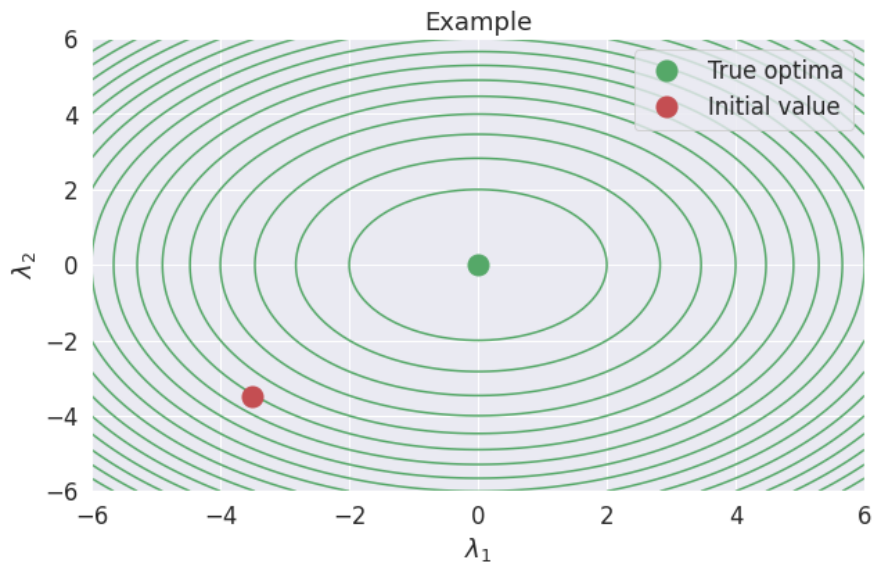
$$p(\mathbf{y}, \mathbf{w}) = \prod_{n=1}^{N} p(y_n|\mathbf{x}_n, \mathbf{w})\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

Black-box variational inference: A few words on stochastic optimization

Simple example: Optimizing $J(\boldsymbol{\lambda})$

- Suppose $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\lambda}, \boldsymbol{I})$ and $\log p(\boldsymbol{y}, \boldsymbol{w}) = -\boldsymbol{w}^T\boldsymbol{w}$

$$J(\boldsymbol{\lambda}) = \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] = \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[-\boldsymbol{w}^T\boldsymbol{w}\right]$$



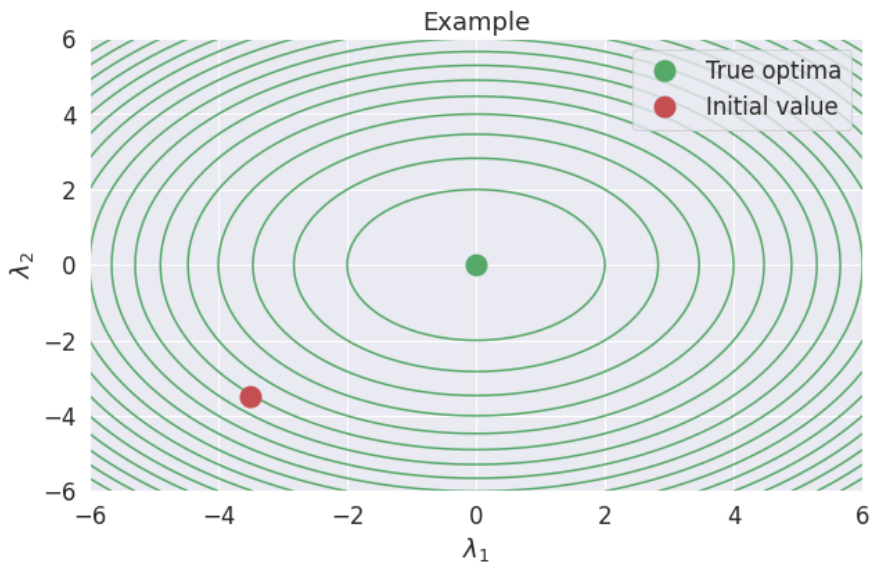
Simple example: Optimizing $J(\boldsymbol{\lambda})$

- Suppose $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\lambda}, \boldsymbol{I})$ and $\log p(\boldsymbol{y}, \boldsymbol{w}) = -\boldsymbol{w}^T\boldsymbol{w}$

$$J(\boldsymbol{\lambda}) = \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] = \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[-\boldsymbol{w}^T\boldsymbol{w}\right]$$

- We optimize J using gradient ascent with step-size $\rho = 0.1$

$$\boldsymbol{\lambda}^{t+1} = \boldsymbol{\lambda}^t + \rho\hat{g}(\boldsymbol{\lambda}^t)$$



Simple example: Optimizing $J(\boldsymbol{\lambda})$

- Suppose $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\lambda}, \boldsymbol{I})$ and $\log p(\boldsymbol{y}, \boldsymbol{w}) = -\boldsymbol{w}^T\boldsymbol{w}$

$$J(\boldsymbol{\lambda}) = \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}, \boldsymbol{w})] = \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[-\boldsymbol{w}^T\boldsymbol{w}\right]$$

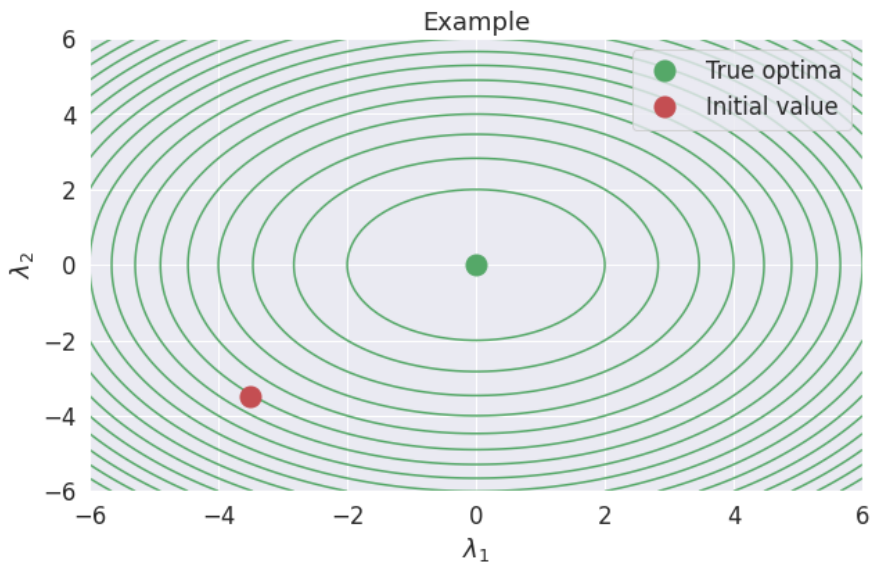
- We optimize J using gradient ascent with step-size $\rho = 0.1$

$$\boldsymbol{\lambda}^{t+1} = \boldsymbol{\lambda}^t + \rho\hat{g}(\boldsymbol{\lambda}^t)$$

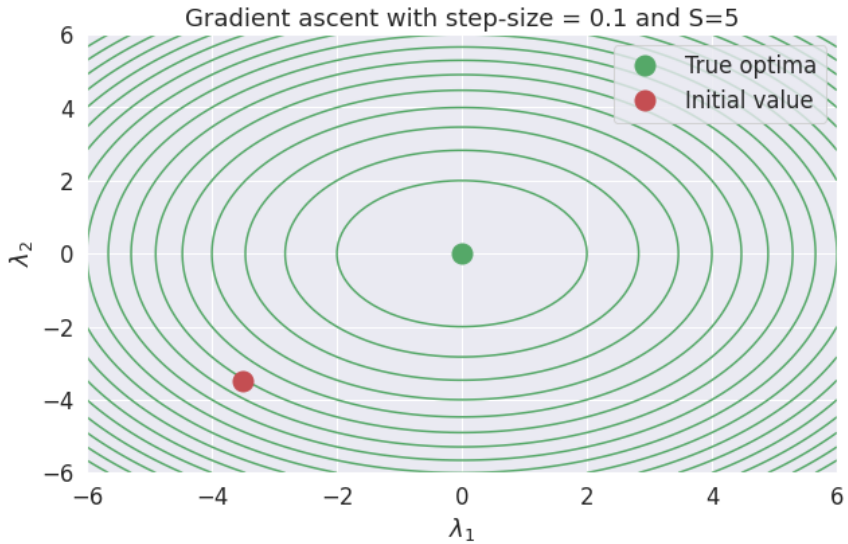
- Using the score function gradient estimator

$$\hat{g} = \frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log q_{\boldsymbol{\lambda}}(\boldsymbol{w}_s) \log p(\boldsymbol{y}, \boldsymbol{w}_s) = \frac{1}{S}\sum_{s=1}^{S} 2\left(\boldsymbol{w}_s - \boldsymbol{\lambda}\right)\boldsymbol{w}_s^T\boldsymbol{w}_s$$

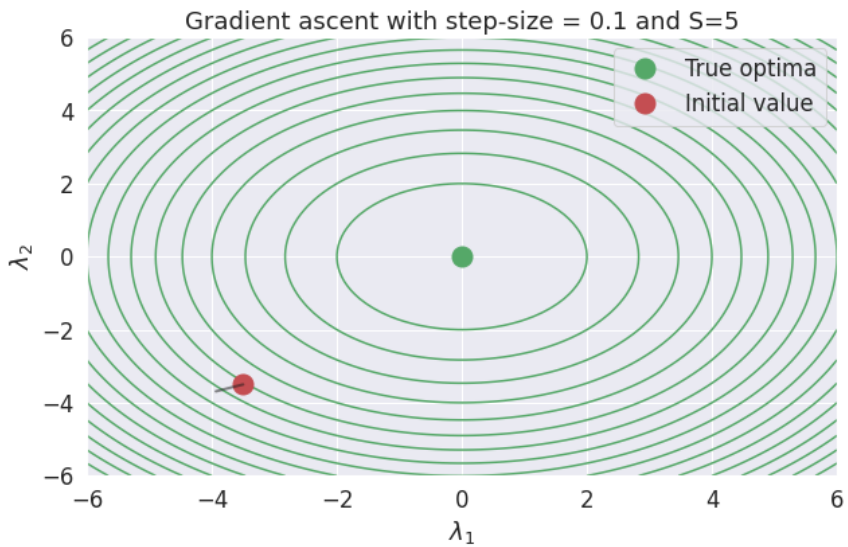
where $\boldsymbol{w}_s \sim q_{\boldsymbol{\lambda}}(\boldsymbol{w})$



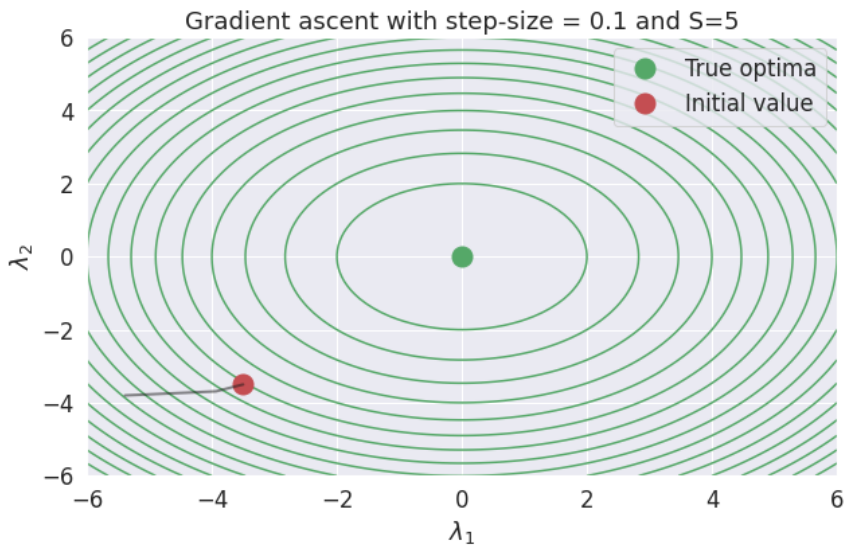
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



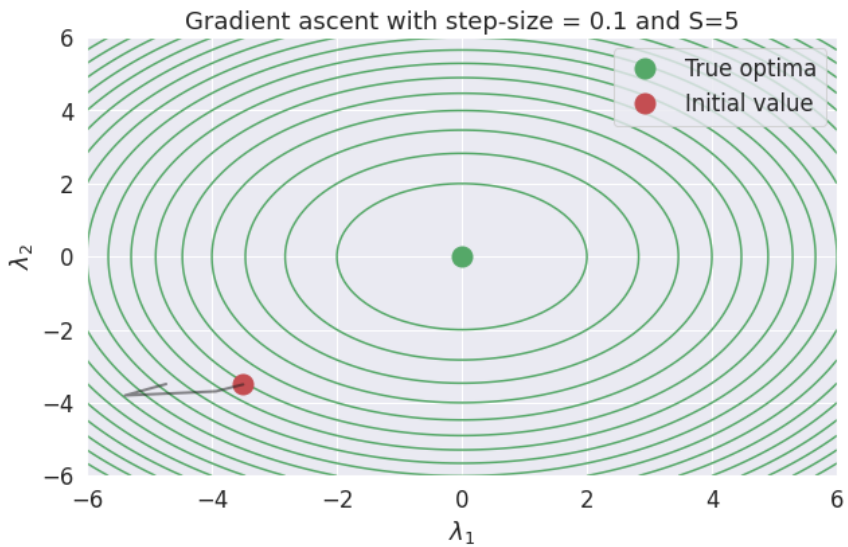
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



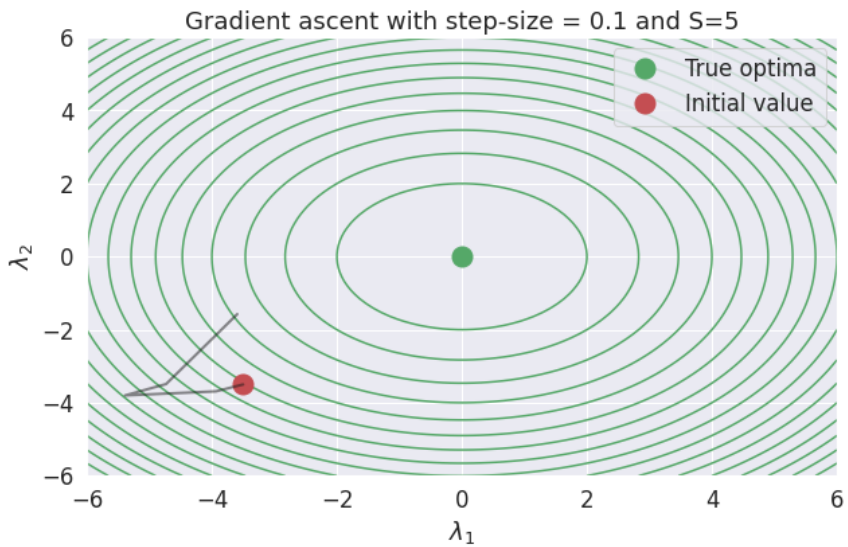
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



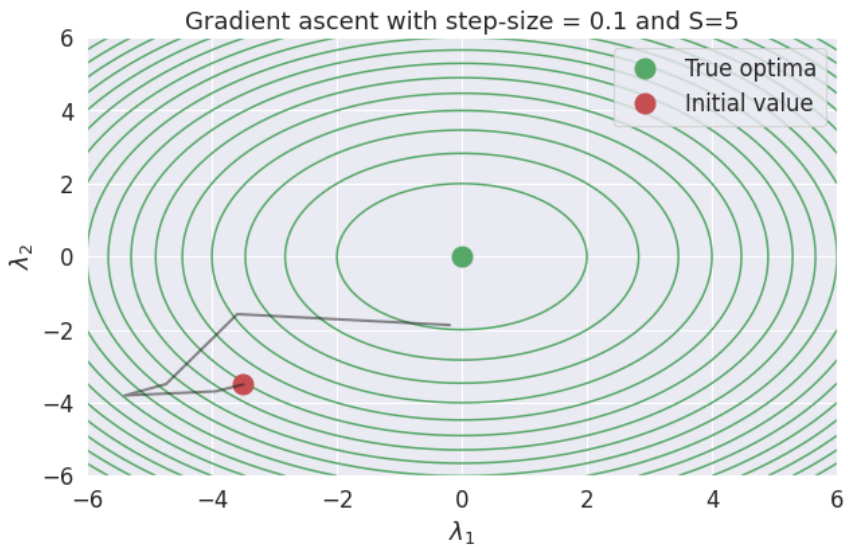
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



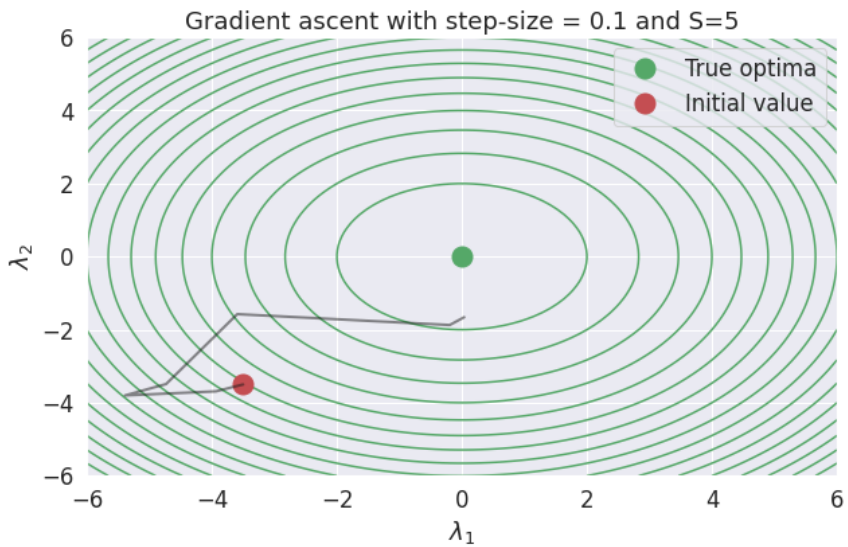
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



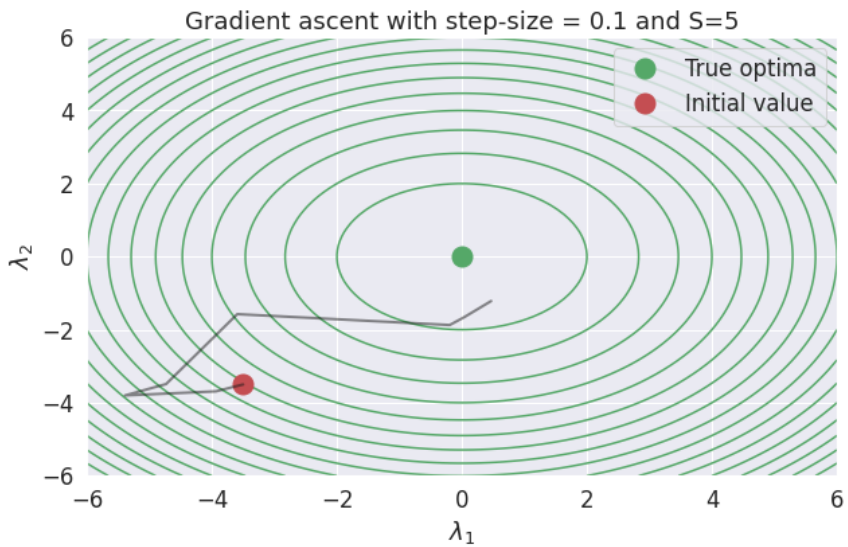
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



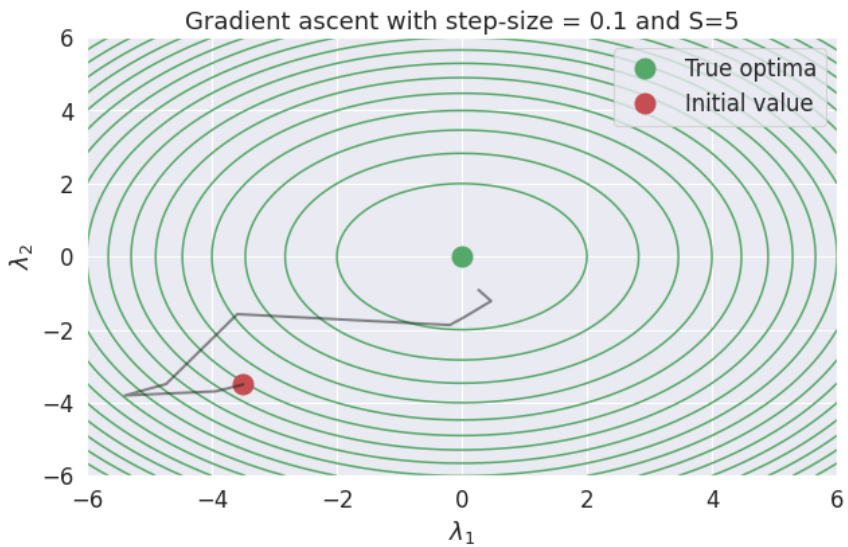
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



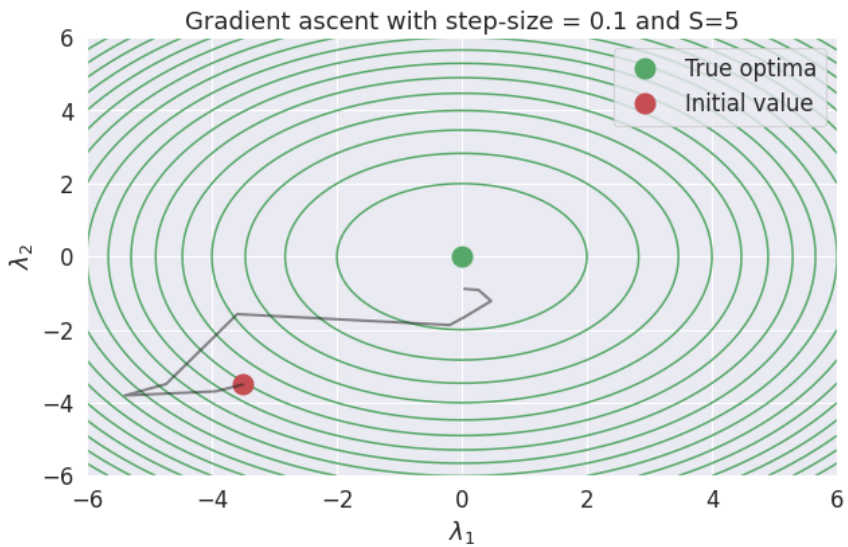
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



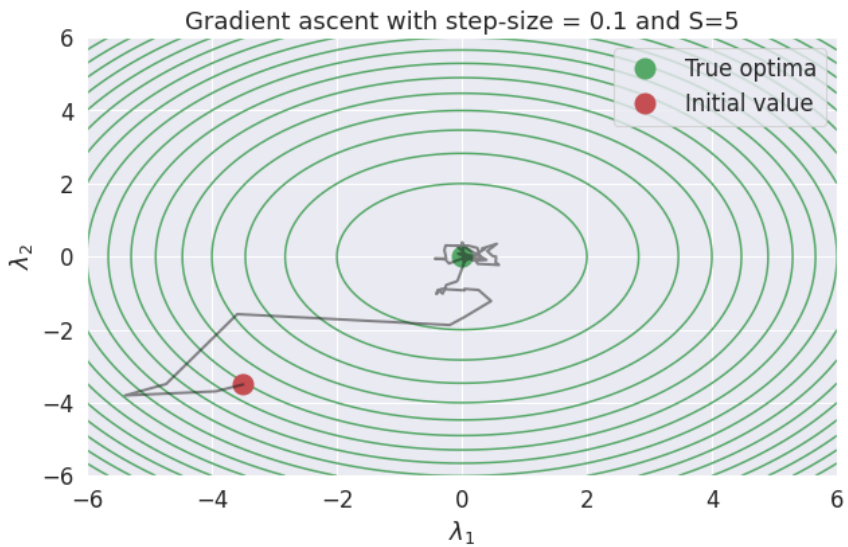
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



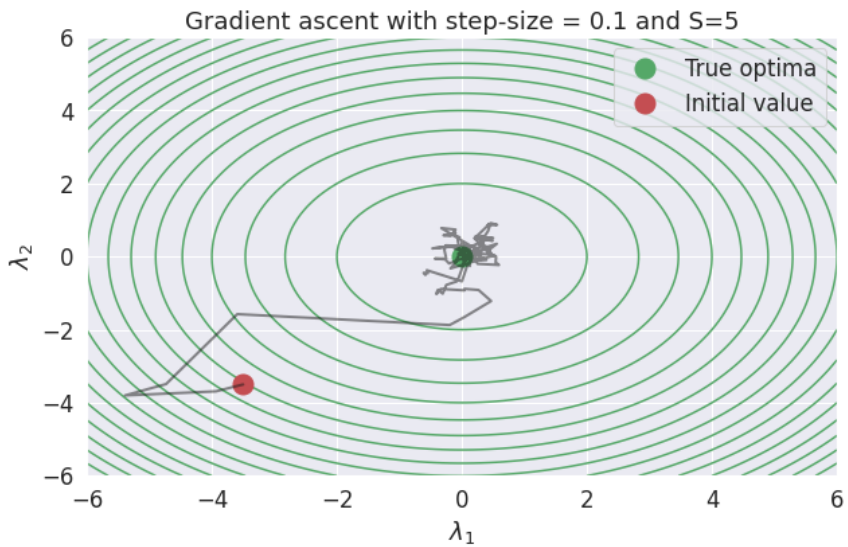
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



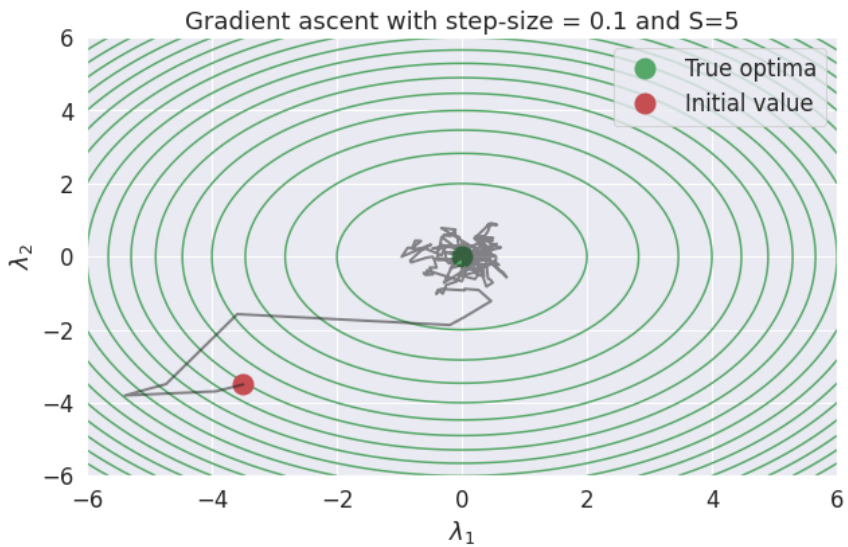
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II



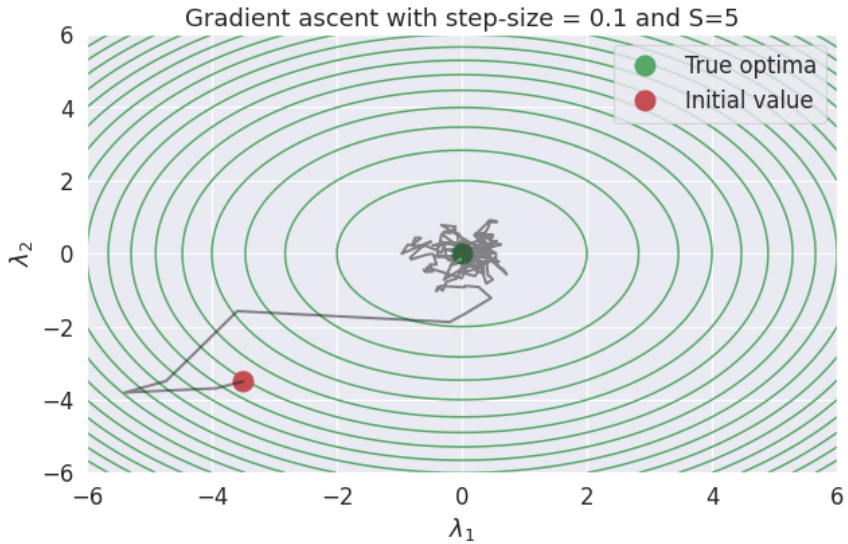
Simple example: Optimizing $J(\boldsymbol{\lambda})$ II

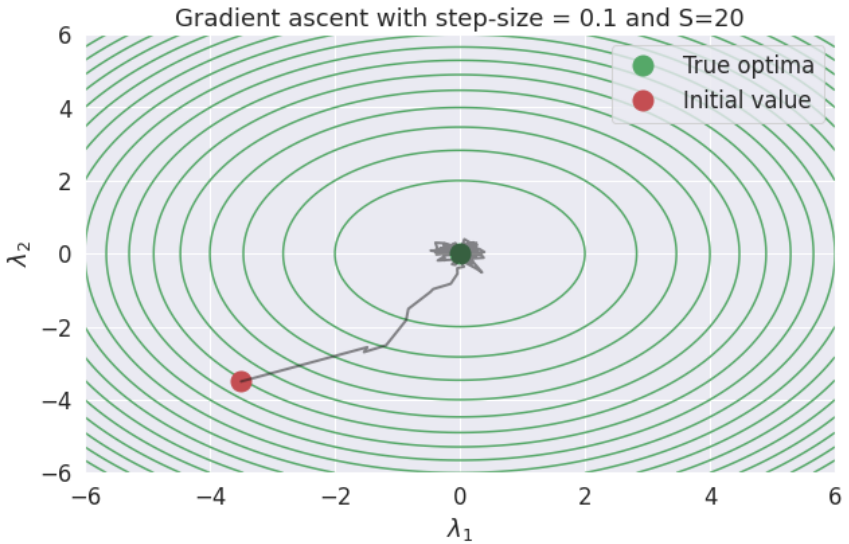


Simple example: Optimizing $J(\boldsymbol{\lambda})$ II

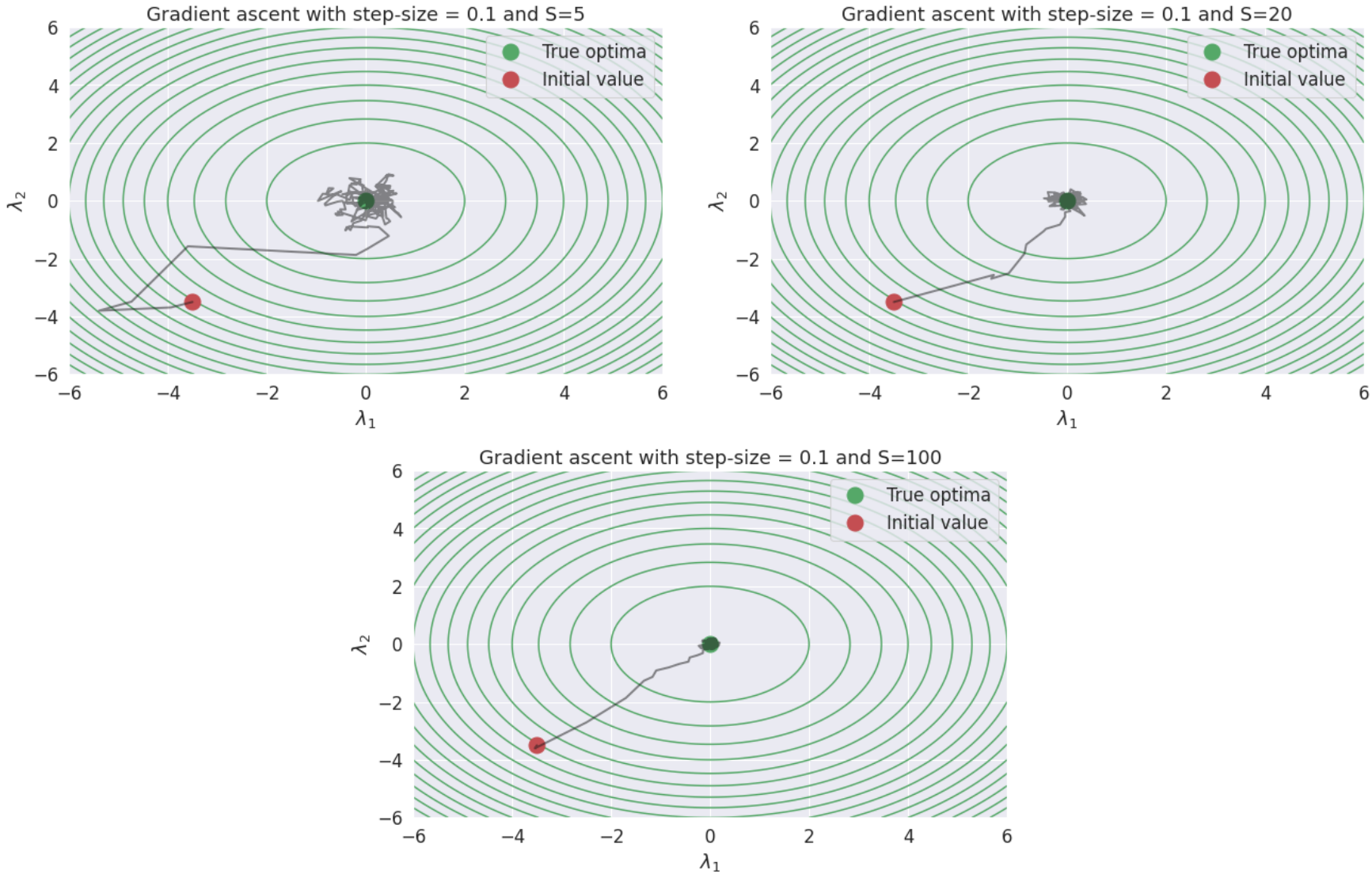


Simple example: Optimizing $J(\boldsymbol{\lambda})$ II

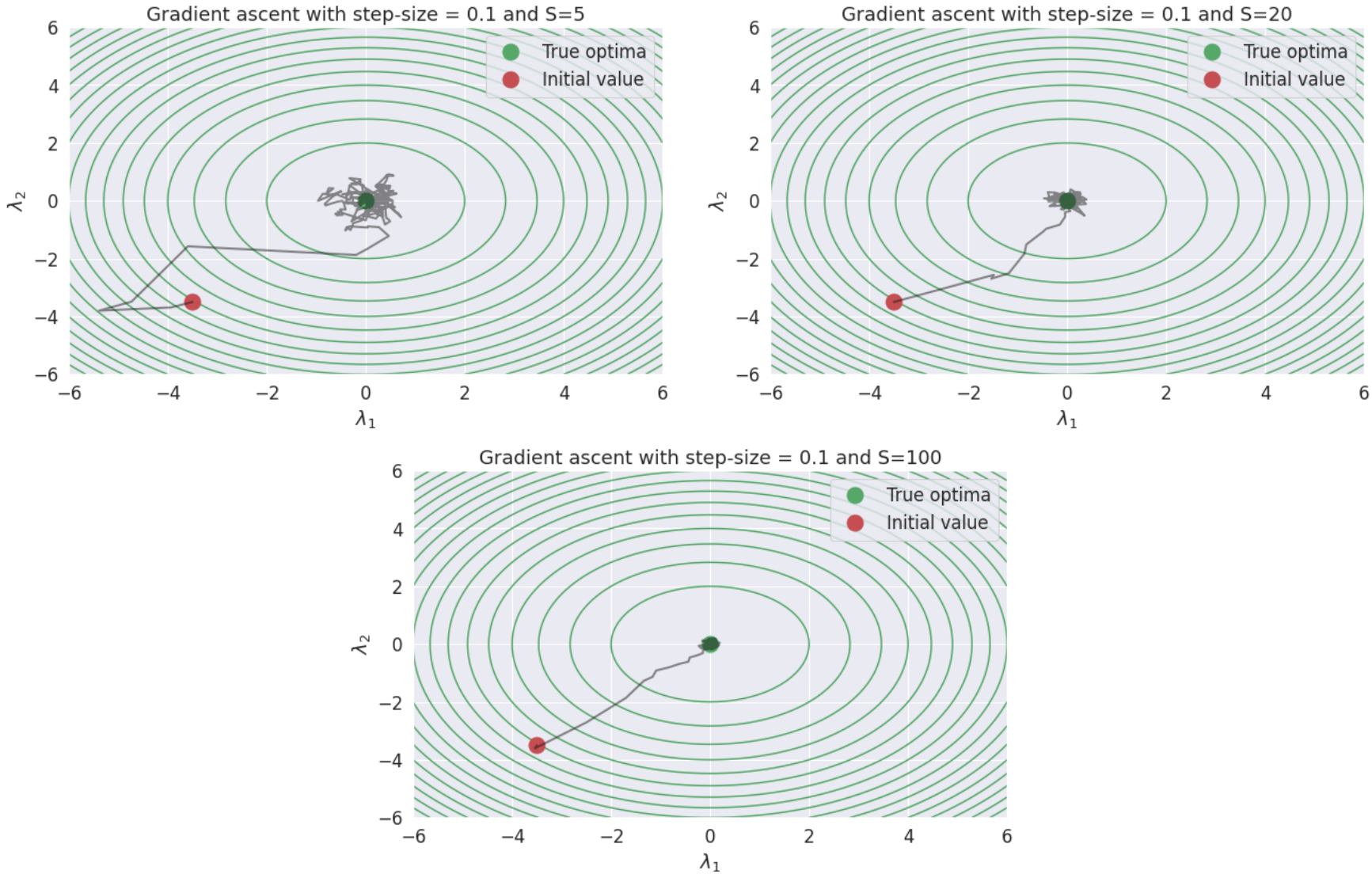




Simple example: Optimizing $J(\lambda)$ II

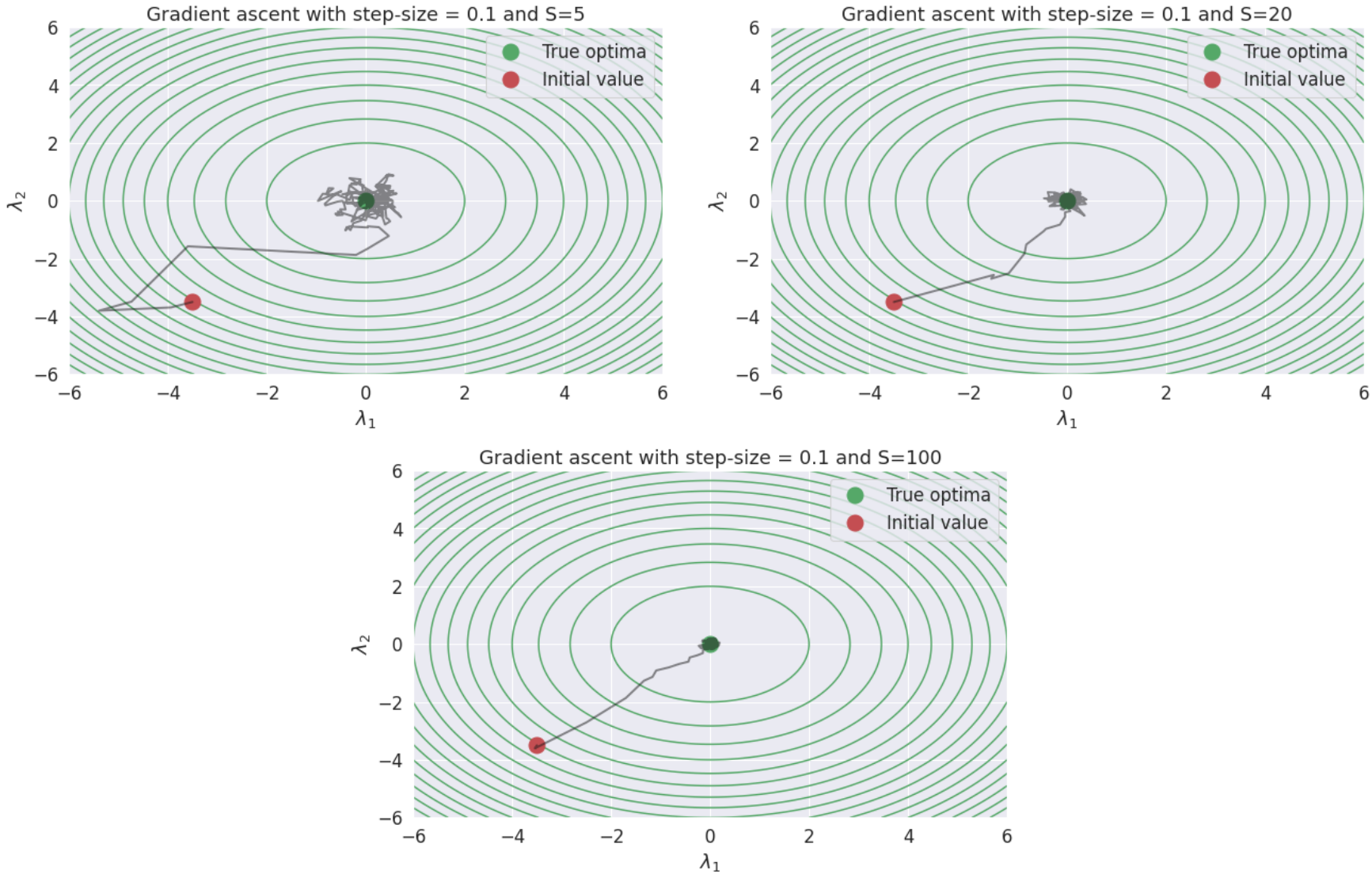


Simple example: Optimizing $J(\lambda)$ II



- Gradient ascent does not converge with constant step size

Simple example: Optimizing $J(\lambda)$ II



- Gradient ascent does not converge with constant step size
- The number of samples S trades off gradient variance vs computational cost

Stochastic gradient descent (SGD)

- The gradient ascent update with *constant step-size* η

$$\boldsymbol{\lambda}^{t+1} = \boldsymbol{\lambda}^{t} + \eta \hat{g}(\boldsymbol{\lambda}^{t})$$

- In *stochastic gradient ascent* the step-size is decreased in each iterations

$$\boldsymbol{\lambda}^{t+1} = \boldsymbol{\lambda}^{t} + \rho_t \eta \hat{g}(\boldsymbol{\lambda}^{t})$$

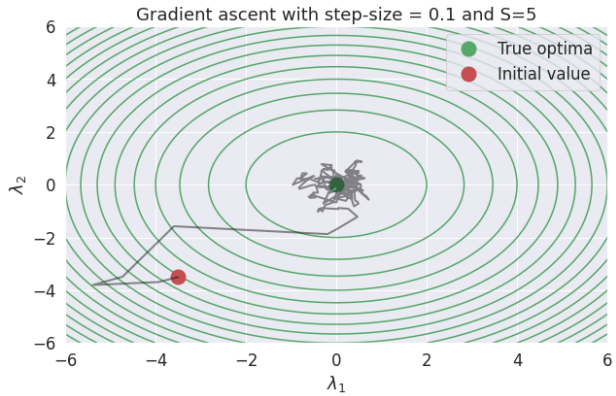
- The *Robbins-Monro conditions* guarentees convergence in probability if the gradient estimator is unbiased (and under regularity conditions)

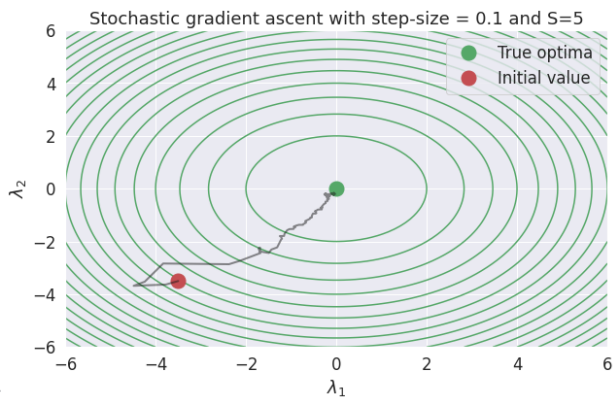
$$\sum_{t=1}^{\infty} \rho_t = \infty \qquad \text{and} \qquad \sum_{t=1}^{\infty} \rho_t^2 \leq \infty$$

- Example: $\tau > 0$ and $\kappa \in (0.5, 1]$

$$\rho_t = (t + \tau)^{-\kappa}$$

Robbins & Monro (1951): A stochastic approximation method





A zoo of stochastic optimizers

- Many recent methods for stochastic optimization: Adam, RMSProp, AdaGrad, AdaDelta etc
- Several key ideas for faster and more robust optimization

Kingma & Ba (2014): Adam: A Method for Stochastic Optimization (70k+ citations in 2021, 170k+ today)

A zoo of stochastic optimizers

- Many recent methods for stochastic optimization: Adam, RMSProp, AdaGrad, AdaDelta etc
- Several key ideas for faster and more robust optimization
- *Momentum*: use gradient from previous iteration (rolling ball analogy)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$\lambda_{t+1} = \lambda_t + \eta m_t$$

Kingma & Ba (2014): Adam: A Method for Stochastic Optimization (70k+ citations in 2021, 170k+ today)

A zoo of stochastic optimizers

- Many recent methods for stochastic optimization: Adam, RMSProp, AdaGrad, AdaDelta etc
- Several key ideas for faster and more robust optimization
- *Momentum*: use gradient from previous iteration (rolling ball analogy)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$\lambda_{t+1} = \lambda_t + \eta m_t$$

- *Adaptive learning rates*: Adaptive learning rate based on magnitude of gradient

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
$$\lambda_{t+1} = \lambda_t + \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Kingma & Ba (2014): Adam: A Method for Stochastic Optimization (70k+ citations in 2021, 170k+ today)

A zoo of stochastic optimizers

- Many recent methods for stochastic optimization: Adam, RMSProp, AdaGrad, AdaDelta etc
- Several key ideas for faster and more robust optimization
- *Momentum*: use gradient from previous iteration (rolling ball analogy)

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$\lambda_{t+1} = \lambda_t + \eta m_t$$

- *Adaptive learning rates*: Adaptive learning rate based on magnitude of gradient

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\lambda_{t+1} = \lambda_t + \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

- *Individual learning*: Each parameter is allowed to have it is adaptive learning rate

Kingma & Ba (2014): Adam: A Method for Stochastic Optimization (70k+ citations in 2021, 170k+ today)

A zoo of stochastic optimizers

- Many recent methods for stochastic optimization: Adam, RMSProp, AdaGrad, AdaDelta etc
- Several key ideas for faster and more robust optimization
- *Momentum*: use gradient from previous iteration (rolling ball analogy)

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$\lambda_{t+1} = \lambda_t + \eta m_t$$

- *Adaptive learning rates*: Adaptive learning rate based on magnitude of gradient

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\lambda_{t+1} = \lambda_t + \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

- *Individual learning*: Each parameter is allowed to have it is adaptive learning rate
- The ADAM optimizer combines all these ideas (along with a bias-correction) and is often a good first choice

Kingma & Ba (2014): Adam: A Method for Stochastic Optimization (70k+ citations in 2021, 170k+ today)

A zoo of stochastic optimizers

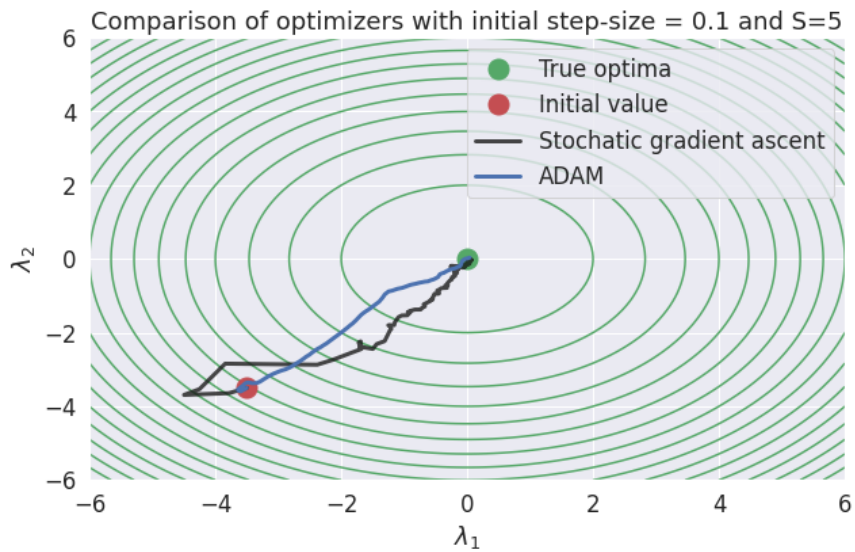
- Many recent methods for stochastic optimization: Adam, RMSProp, AdaGrad, AdaDelta etc
- Several key ideas for faster and more robust optimization
- *Momentum*: use gradient from previous iteration (rolling ball analogy)

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$\lambda_{t+1} = \lambda_t + \eta m_t$$

- *Adaptive learning rates*: Adaptive learning rate based on magnitude of gradient

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\lambda_{t+1} = \lambda_t + \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

- *Individual learning*: Each parameter is allowed to have it is adaptive learning rate
- The ADAM optimizer combines all these ideas (along with a bias-correction) and is often a good first choice



Kingma & Ba (2014): Adam: A Method for Stochastic Optimization (70k+ citations in 2021, 170k+ today)

Quiz time

Quiz time!

DTU Learn - Quiz - Lecture 12 - BBVI quiz

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

- If $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$ is Gaussian and if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{w} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$, where $\boldsymbol{L}$ is the Cholesky factorization (matrix square root) of $\boldsymbol{V} = \boldsymbol{L}\boldsymbol{L}^T$.

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

- If $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$ is Gaussian and if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{w} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$, where $\boldsymbol{L}$ is the Cholesky factorization (matrix square root) of $\boldsymbol{V} = \boldsymbol{L}\boldsymbol{L}^T$.

- We *re-parametrize* $\boldsymbol{w}$ as $\boldsymbol{w}_\epsilon = g(\boldsymbol{\lambda}, \boldsymbol{\epsilon}) = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon}$

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log p(\boldsymbol{y}, \boldsymbol{w})]$$

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

- If $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$ is Gaussian and if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{w} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$, where $\boldsymbol{L}$ is the Cholesky factorization (matrix square root) of $\boldsymbol{V} = \boldsymbol{L}\boldsymbol{L}^T$.

- We *re-parametrize* $\boldsymbol{w}$ as $\boldsymbol{w}_\epsilon = g(\boldsymbol{\lambda}, \boldsymbol{\epsilon}) = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon}$

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log p(\boldsymbol{y}, \boldsymbol{w})] = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon)] \qquad \text{(Re-parametrization)}$$

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

- If $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$ is Gaussian and if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{w} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$, where $\boldsymbol{L}$ is the Cholesky factorization (matrix square root) of $\boldsymbol{V} = \boldsymbol{L}\boldsymbol{L}^T$.

- We *re-parametrize* $\boldsymbol{w}$ as $\boldsymbol{w}_\epsilon = g(\boldsymbol{\lambda}, \boldsymbol{\epsilon}) = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon}$

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log p(\boldsymbol{y}, \boldsymbol{w})] = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon)] \qquad \text{(Re-parametrization)}$$

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

- If $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$ is Gaussian and if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{w} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$, where $\boldsymbol{L}$ is the Cholesky factorization (matrix square root) of $\boldsymbol{V} = \boldsymbol{L}\boldsymbol{L}^T$.

- We *re-parametrize* $\boldsymbol{w}$ as $\boldsymbol{w}_\epsilon = g(\boldsymbol{\lambda}, \boldsymbol{\epsilon}) = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon}$

$$\begin{aligned} \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log p(\boldsymbol{y}, \boldsymbol{w})] &= \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon)] && \text{(Re-parametrization)} \\ &= \nabla_{\boldsymbol{\lambda}} \int q(\boldsymbol{\epsilon}) \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon) \mathrm{d}\boldsymbol{\epsilon} && \text{(Def. of expectation)} \end{aligned}$$

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

- If $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$ is Gaussian and if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{w} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$, where $\boldsymbol{L}$ is the Cholesky factorization (matrix square root) of $\boldsymbol{V} = \boldsymbol{L}\boldsymbol{L}^T$.

- We *re-parametrize* $\boldsymbol{w}$ as $\boldsymbol{w}_\epsilon = g(\boldsymbol{\lambda}, \boldsymbol{\epsilon}) = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon}$

$$
\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log p(\boldsymbol{y}, \boldsymbol{w})] &= \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon)] && \text{(Re-parametrization)} \\
&= \nabla_{\boldsymbol{\lambda}} \int q(\boldsymbol{\epsilon}) \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon) \mathrm{d}\boldsymbol{\epsilon} && \text{(Def. of expectation)} \\
&= \int q(\boldsymbol{\epsilon}) \nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon) \mathrm{d}\boldsymbol{\epsilon} && \text{(Leibniz' rule)}
\end{aligned}
$$

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

- If $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$ is Gaussian and if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{w} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$, where $\boldsymbol{L}$ is the Cholesky factorization (matrix square root) of $\boldsymbol{V} = \boldsymbol{L}\boldsymbol{L}^T$.

- We *re-parametrize* $\boldsymbol{w}$ as $\boldsymbol{w}_\epsilon = g(\boldsymbol{\lambda}, \boldsymbol{\epsilon}) = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon}$

$$
\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log p(\boldsymbol{y}, \boldsymbol{w})] &= \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon)] && \text{(Re-parametrization)} \\
&= \nabla_{\boldsymbol{\lambda}} \int q(\boldsymbol{\epsilon}) \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon) \mathrm{d}\boldsymbol{\epsilon} && \text{(Def. of expectation)} \\
&= \int q(\boldsymbol{\epsilon}) \nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon) \mathrm{d}\boldsymbol{\epsilon} && \text{(Leibniz' rule)} \\
&= \mathbb{E}_{q(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon)] && \text{(Def. of expectation)}
\end{aligned}
$$

The re-parametrization trick: gradient estimator with lower variance

- Gradient estimators with lower variance leads to faster optimization, because *higher variance requires smaller step-sizes*. Often we can do better than the score function gradient

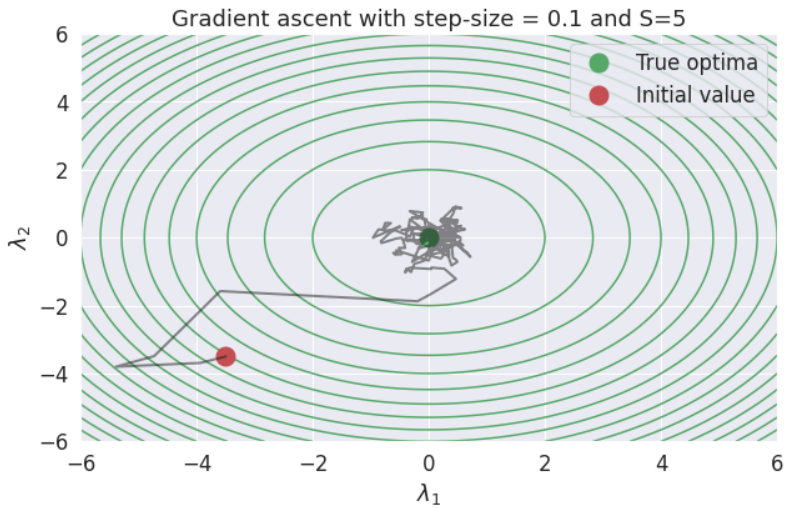
- If $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$ is Gaussian and if $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{w} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \mathbf{V})$, where $\boldsymbol{L}$ is the Cholesky factorization (matrix square root) of $\boldsymbol{V} = \boldsymbol{L}\boldsymbol{L}^T$.

- We *re-parametrize* $\boldsymbol{w}$ as $\boldsymbol{w}_\epsilon = g(\boldsymbol{\lambda}, \boldsymbol{\epsilon}) = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{\epsilon}$

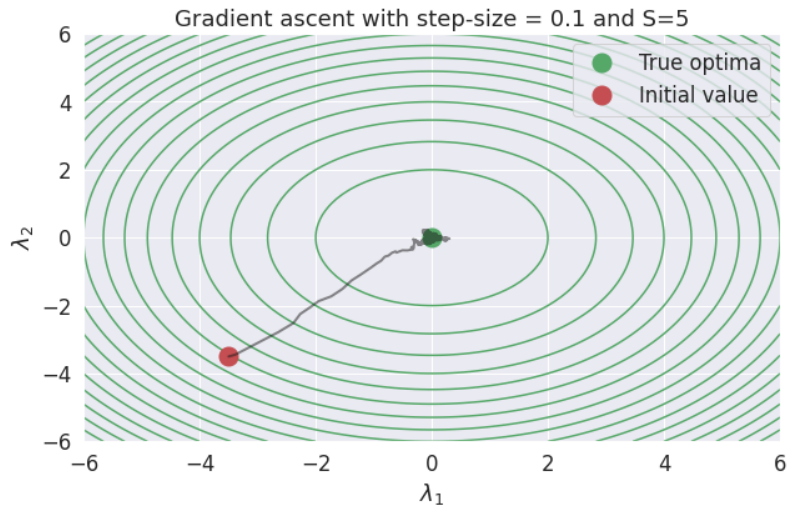
$$
\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log p(\boldsymbol{y}, \boldsymbol{w})] &= \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon)] && \text{(Re-parametrization)} \\
&= \nabla_{\boldsymbol{\lambda}} \int q(\boldsymbol{\epsilon}) \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon) \mathrm{d}\boldsymbol{\epsilon} && \text{(Def. of expectation)} \\
&= \int q(\boldsymbol{\epsilon}) \nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon) \mathrm{d}\boldsymbol{\epsilon} && \text{(Leibniz' rule)} \\
&= \mathbb{E}_{q(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_\epsilon)] && \text{(Def. of expectation)} \\
&\approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) \qquad \text{where} \qquad \boldsymbol{\epsilon}_s \sim q(\boldsymbol{\epsilon})
\end{aligned}
$$

- The *re-parametrized gradient estimator* requires the transformation g and $p(\boldsymbol{y}, \boldsymbol{w})$ to be differentiable, which rules out discrete variables.

Simple example revisited



Score function gradient estimator



Re-parametrized gradient estimator

- Comparison of the score function gradient estimator and the re-parametrized gradient estimator
- Using gradient ascent with fixed step-size for the purpose of illustration
- Score function is more general, but suffers from larger variance
- The re-parametrized gradient is less general and applies only to continuous variables, but has much lower variance

Reparametrized gradients

- Many variational families can be re-parametrized

Target	$p(z;\theta)$	Base $p(\epsilon)$	One-liner $g(\epsilon;\theta)$
Exponential	$\exp(-x); x > 0$	$\epsilon \sim [0;1]$	$\ln(1/\epsilon)$
Cauchy	$\frac{1}{\pi(1+x^2)}$	$\epsilon \sim [0;1]$	$\tan(\pi\epsilon)$
Laplace	$\mathcal{L}(0;1) = \exp(-\lvert x\rvert)$	$\epsilon \sim [0;1]$	$\ln(\frac{\epsilon_1}{\epsilon_2})$
Laplace	$\mathcal{L}(\mu;b)$	$\epsilon \sim [0;1]$	$\mu - bsgn(\epsilon)\ln(1-2\lvert\epsilon\rvert)$
Std Gaussian	$\mathcal{N}(0;1)$	$\epsilon \sim [0;1]$	$\sqrt{\ln(\frac{1}{\epsilon_1})}\cos(2\pi\epsilon_2)$
Gaussian	$\mathcal{N}(\mu;RR^\top)$	$\epsilon \sim \mathcal{N}(0;1)$	$\mu + R\epsilon$
Rademacher	$Rad(\frac{1}{2})$	$\epsilon \sim Bern(\frac{1}{2})$	$2\epsilon - 1$
Log-Normal	$\ln \mathcal{N}(\mu;\sigma)$	$\epsilon \sim \mathcal{N}(\mu;\sigma^2)$	$\exp(\epsilon)$
Inv Gamma	$i\mathcal{G}(k;\theta)$	$\epsilon \sim \mathcal{G}(k;\theta^{-1})$	$\frac{1}{\epsilon}$

Shakir Mohamed: `http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/`

Black-box variational inference: Mini-batching for large scale inference

Mini-batching for large scale inference

- Assuming a model of the form

$$p(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{w})p(\boldsymbol{w})$$

Mini-batching for large scale inference

- Assuming a model of the form

$$p(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{w})p(\boldsymbol{w})$$

- Let $\boldsymbol{w}^s \sim q_{\boldsymbol{\lambda}}$ be samples from the variational approximation, then

$$\begin{aligned}\mathcal{L}[q] &\approx \frac{1}{S}\sum_{s=1}^{S}\left[\log p(\boldsymbol{y}, \boldsymbol{w}^s)\right] + \mathcal{H}[q] \\ &= \frac{1}{S}\sum_{s=1}^{S}\left[\sum_{n=1}^{N}\log p(y_n, \boldsymbol{w}^s) + \log p(\boldsymbol{w}^s)\right] + \mathcal{H}[q]\end{aligned}$$

Mini-batching for large scale inference

- Assuming a model of the form

$$p(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{w})p(\boldsymbol{w})$$

- Let $\boldsymbol{w}^s \sim q_{\boldsymbol{\lambda}}$ be samples from the variational approximation, then

$$\begin{aligned}\mathcal{L}[q] &\approx \frac{1}{S}\sum_{s=1}^{S}\left[\log p(\boldsymbol{y}, \boldsymbol{w}^s)\right] + \mathcal{H}[q] \\ &= \frac{1}{S}\sum_{s=1}^{S}\left[\sum_{n=1}^{N}\log p(y_n, \boldsymbol{w}^s) + \log p(\boldsymbol{w}^s)\right] + \mathcal{H}[q]\end{aligned}$$

- Evaluating the ELBO and its gradients becomes slow for large N

Mini-batching for large scale inference

- Assuming a model of the form

$$p(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{w})p(\boldsymbol{w})$$

- Let $\boldsymbol{w}^s \sim q_{\boldsymbol{\lambda}}$ be samples from the variational approximation, then

$$\begin{aligned}\mathcal{L}[q] &\approx \frac{1}{S}\sum_{s=1}^{S}\left[\log p(\boldsymbol{y}, \boldsymbol{w}^s)\right] + \mathcal{H}[q] \\ &= \frac{1}{S}\sum_{s=1}^{S}\left[\sum_{n=1}^{N}\log p(y_n, \boldsymbol{w}^s) + \log p(\boldsymbol{w}^s)\right] + \mathcal{H}[q]\end{aligned}$$

- Evaluating the ELBO and its gradients becomes slow for large N

- *Mini-batching:* Let $\mathcal{M}$ be a random sample of the data points of size M, then the following is an unbiased estimator of the ELBO

$$\mathcal{L}[q] \approx \frac{1}{S}\sum_{s=1}^{S}\left[\frac{N}{M}\sum_{n\in\mathcal{M}}\log p(y_n, \boldsymbol{w}^s) + \log p(\boldsymbol{w}^s)\right] + \mathcal{H}[q]$$

Mini-batching for large scale inference

- Assuming a model of the form

$$p(\boldsymbol{y}, \boldsymbol{w}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{w})p(\boldsymbol{w})$$

- Let $\boldsymbol{w}^s \sim q_{\boldsymbol{\lambda}}$ be samples from the variational approximation, then

$$\begin{aligned}\mathcal{L}[q] &\approx \frac{1}{S}\sum_{s=1}^{S}\left[\log p(\boldsymbol{y}, \boldsymbol{w}^s)\right] + \mathcal{H}[q] \\ &= \frac{1}{S}\sum_{s=1}^{S}\left[\sum_{n=1}^{N}\log p(y_n, \boldsymbol{w}^s) + \log p(\boldsymbol{w}^s)\right] + \mathcal{H}[q]\end{aligned}$$

- Evaluating the ELBO and its gradients becomes slow for large N

- *Mini-batching:* Let $\mathcal{M}$ be a random sample of the data points of size M, then the following is an unbiased estimator of the ELBO

$$\mathcal{L}[q] \approx \frac{1}{S}\sum_{s=1}^{S}\left[\frac{N}{M}\sum_{n\in\mathcal{M}}\log p(y_n, \boldsymbol{w}^s) + \log p(\boldsymbol{w}^s)\right] + \mathcal{H}[q]$$

- We can use the same ideas to speed up the calculations of the gradients, but the price is increased variance

Summary

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. the re-parametrized gradient

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$

Summary

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. the re-parametrized gradient

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)

Summary

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. the re-parametrized gradient

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{\epsilon}_s \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \boldsymbol{I})$ for $s = 1, \ldots, S$

Summary

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. the re-parametrized gradient

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{\epsilon}_s \sim \mathcal{N}(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$ for $s = 1, \ldots, S$
  2. Estimate ELBO (use mini-batching if necessary)

$$\hat{\mathcal{L}}\left[q\right] = \frac{1}{S}\sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \mathcal{H}\left[q\right], \qquad \text{where} \quad \boldsymbol{w}_{\epsilon_s} = \boldsymbol{m}_t + \boldsymbol{L}_t\epsilon_s$$

Summary

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. the re-parametrized gradient

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{\epsilon}_s \sim \mathcal{N}(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$ for $s = 1, \ldots, S$
  2. Estimate ELBO (use mini-batching if necessary)

$$\hat{\mathcal{L}}\left[q\right] = \frac{1}{S}\sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \mathcal{H}\left[q\right], \qquad \text{where} \quad \boldsymbol{w}_{\epsilon_s} = \boldsymbol{m}_t + \boldsymbol{L}_t\epsilon_s$$

  3. Estimate gradient (use mini-batching if necessary)

$$\nabla_{\boldsymbol{\lambda}}\hat{\mathcal{L}}\left[q\right] \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \nabla_{\boldsymbol{\lambda}}\mathcal{H}\left[q\right]$$

Summary

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. the re-parametrized gradient

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{\epsilon}_s \sim \mathcal{N}(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$ for $s = 1, \ldots, S$
  2. Estimate ELBO (use mini-batching if necessary)

$$\hat{\mathcal{L}}\left[q\right] = \frac{1}{S}\sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \mathcal{H}\left[q\right], \qquad \text{where} \quad \boldsymbol{w}_{\epsilon_s} = \boldsymbol{m}_t + \boldsymbol{L}_t\epsilon_s$$

  3. Estimate gradient (use mini-batching if necessary)

$$\nabla_{\boldsymbol{\lambda}}\hat{\mathcal{L}}\left[q\right] \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \nabla_{\boldsymbol{\lambda}}\mathcal{H}\left[q\right]$$

  4. Update variational parameters to get $\boldsymbol{\lambda}_{t+1} = \{\boldsymbol{m}_{t+1}, \log \boldsymbol{v}_{t+1}\}$ using gradient estimate

Summary

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. the re-parametrized gradient

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{\epsilon}_s \sim \mathcal{N}(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$ for $s = 1, \ldots, S$
  2. Estimate ELBO (use mini-batching if necessary)

$$\hat{\mathcal{L}}[q] = \frac{1}{S}\sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \mathcal{H}[q], \qquad \text{where} \quad \boldsymbol{w}_{\epsilon_s} = \boldsymbol{m}_t + \boldsymbol{L}_t\epsilon_s$$

  3. Estimate gradient (use mini-batching if necessary)

$$\nabla_{\boldsymbol{\lambda}}\hat{\mathcal{L}}[q] \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \nabla_{\boldsymbol{\lambda}}\mathcal{H}[q]$$

  4. Update variational parameters to get $\boldsymbol{\lambda}_{t+1} = \{\boldsymbol{m}_{t+1}, \log \boldsymbol{v}_{t+1}\}$ using gradient estimate
- Predictions & model checking: Evaluate training error, validation errors etc

Summary

Putting everything together

Given a model and variational family (e.g. mean-field Gaussians)

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \qquad q(\boldsymbol{w}) = \prod_{i=1}^{D} \mathcal{N}(w_i|m_i, v_i)$$

Black-box variational inference w. the re-parametrized gradient

- Initialize variational parameters $\boldsymbol{\lambda}_1 = \{\boldsymbol{m}_1, \log \boldsymbol{v}_1\}$
- For iteration $t = 1, \ldots, T$ (or until convergence)
  1. Generate samples $\boldsymbol{\epsilon}_s \sim \mathcal{N}(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$ for $s = 1, \ldots, S$
  2. Estimate ELBO (use mini-batching if necessary)

$$\hat{\mathcal{L}}[q] = \frac{1}{S}\sum_{s=1}^{S} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \mathcal{H}[q], \qquad \text{where} \quad \boldsymbol{w}_{\epsilon_s} = \boldsymbol{m}_t + \boldsymbol{L}_t\epsilon_s$$

  3. Estimate gradient (use mini-batching if necessary)

$$\nabla_{\boldsymbol{\lambda}}\hat{\mathcal{L}}[q] \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\lambda}} \log p(\boldsymbol{y}, \boldsymbol{w}_{\epsilon_s}) + \nabla_{\boldsymbol{\lambda}}\mathcal{H}[q]$$

  4. Update variational parameters to get $\boldsymbol{\lambda}_{t+1} = \{\boldsymbol{m}_{t+1}, \log \boldsymbol{v}_{t+1}\}$ using gradient estimate
- Predictions & model checking: Evaluate training error, validation errors etc
- Go back and refine model if needed