Deep learning: inference

Brooks Paige

Week 7

How can we approximate the posterior?

Two basic approaches:

How can we approximate the posterior?

Two basic approaches:

- **Parametric approximation** to the posterior $p(\boldsymbol{\theta}|\mathcal{D})$
  - e.g., a Gaussian distribution $q(\boldsymbol{\theta})$ over weights
  - Variational Bayes, Laplace approximations, ...

How can we approximate the posterior?

Two basic approaches:

- **Parametric approximation** to the posterior $p(\boldsymbol{\theta}|\mathcal{D})$
  - e.g., a Gaussian distribution $q(\boldsymbol{\theta})$ over weights
  - Variational Bayes, Laplace approximations, ...
- **Finite sample approximation** to the posterior
  - e.g., a collection of candidate weights $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$
  - Model ensembles, Markov chain Monte Carlo, Stein variational gradient descent, data augmentation

A brief history of Bayesian deep learning

It's a little hard to trace the history, but roughly:

- Denker & LeCun (1991) optimized the weights of a neural network, and used Laplace's method to approximate a local Gaussian distribution, with the goal of obtaining probabilistic outputs.

A brief history of Bayesian deep learning

It's a little hard to trace the history, but roughly:

- Denker & LeCun (1991) optimized the weights of a neural network, and used Laplace's method to approximate a local Gaussian distribution, with the goal of obtaining probabilistic outputs.
- David MacKay (1992) performs the first systematic study of Bayesian neural networks, using Laplace approximations as a way to perform inference as well as for model comparison.

A brief history of Bayesian deep learning

It's a little hard to trace the history, but roughly:

- Denker & LeCun (1991) optimized the weights of a neural network, and used Laplace's method to approximate a local Gaussian distribution, with the goal of obtaining probabilistic outputs.
- David MacKay (1992) performs the first systematic study of Bayesian neural networks, using Laplace approximations as a way to perform inference as well as for model comparison.
- Hinton & Van Camp (1993), motivated by the "minimum description length" principle, regularize the weights in a network in what is essentially fully-factorized variational inference.

A brief history of Bayesian deep learning

- Radford Neal (1995) developed Markov chain Monte Carlo (MCMC) approaches for drawing posterior samples of the weights, and studies the effect of different prior distributions (and the relationship between Bayesian neural networks and Gaussian processes)

A brief history of Bayesian deep learning

- Radford Neal (1995) developed Markov chain Monte Carlo (MCMC) approaches for drawing posterior samples of the weights, and studies the effect of different prior distributions (and the relationship between Bayesian neural networks and Gaussian processes)
- Williams (1997) provides explicit analytic forms for the relationship between particular Bayesian neural networks and GPs

A brief history of Bayesian deep learning

- Radford Neal (1995) developed Markov chain Monte Carlo (MCMC) approaches for drawing posterior samples of the weights, and studies the effect of different prior distributions (and the relationship between Bayesian neural networks and Gaussian processes)
- Williams (1997) provides explicit analytic forms for the relationship between particular Bayesian neural networks and GPs
- Barber & Bishop (1998) extend variational inference by also modeling correlations between the weights

A brief history of Bayesian deep learning

- Radford Neal (1995) developed Markov chain Monte Carlo (MCMC) approaches for drawing posterior samples of the weights, and studies the effect of different prior distributions (and the relationship between Bayesian neural networks and Gaussian processes)
- Williams (1997) provides explicit analytic forms for the relationship between particular Bayesian neural networks and GPs
- Barber & Bishop (1998) extend variational inference by also modeling correlations between the weights

Then, a large break.

Over a decade later...

- Graves (2011) extends Hinton & Van Camp (1993), using data subsampling, which allows scaling to large datasets

Over a decade later...

- Graves (2011) extends Hinton & Van Camp (1993), using data subsampling, which allows scaling to large datasets
- Blundell et al. (2015) extends this further, introducing reparameterization, a simple algorithm, and alternative priors

Over a decade later...

- Graves (2011) extends Hinton & Van Camp (1993), using data subsampling, which allows scaling to large datasets
- Blundell et al. (2015) extends this further, introducing reparameterization, a simple algorithm, and alternative priors
- Hernández-Lobato & Adams (2015) introduce an alternative method based on expectation propagation

Over a decade later...

- Graves (2011) extends Hinton & Van Camp (1993), using data subsampling, which allows scaling to large datasets
- Blundell et al. (2015) extends this further, introducing reparameterization, a simple algorithm, and alternative priors
- Hernández-Lobato & Adams (2015) introduce an alternative method based on expectation propagation
- Gal & Ghahramani (2015) shows that using Monte Carlo dropout at prediction time approximates the posterior predictive distribution, which is easy to adopt in practice
- Lakshminarayanan et al. (2017) demonstrates that using ensembles of models can provide good predictive uncertainty estimates

Over a decade later...

- Graves (2011) extends Hinton & Van Camp (1993), using data subsampling, which allows scaling to large datasets
- Blundell et al. (2015) extends this further, introducing reparameterization, a simple algorithm, and alternative priors
- Hernández-Lobato & Adams (2015) introduce an alternative method based on expectation propagation
- Gal & Ghahramani (2015) shows that using Monte Carlo dropout at prediction time approximates the posterior predictive distribution, which is easy to adopt in practice
- Lakshminarayanan et al. (2017) demonstrates that using ensembles of models can provide good predictive uncertainty estimates

In recent years, this has become a very popular and crowded area.

Further historical context, if interested

- Great literature review in Section 5 of Wu et al. (2019), *Deterministic Variational Inference for Robust Bayesian Neural Networks.* `https://arxiv.org/abs/1810.03958`

- Yarin' Gal's PhD thesis, Chapter 2

Laplace approximation

Laplace approximations in deep learning

For a posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$, the **Laplace approximation** provides a means for finding an approximate posterior $q(\boldsymbol{\theta})$, which is a Gaussian centered at a posterior mode.

- This is a **local** approximation!
- We're going to go through the same process we went through for linear models, but now with a deep model.

Laplace's method is based on taking a second-order Taylor expansion of the log posterior at the mode $\boldsymbol{\theta}^\star$.

Laplace approximation, in general

In general: suppose we have found the value $\boldsymbol{\theta}^\star$ that maximizes of $\log p(\boldsymbol{\theta}|\mathcal{D})$.

- Take a Taylor expansion of $\log p(\boldsymbol{\theta}|\mathcal{D})$ in the vicinity of $\boldsymbol{\theta}$. For small $\boldsymbol{\delta}$,

$$\log p(\boldsymbol{\theta}^\star + \boldsymbol{\delta}|\mathcal{D}) \approx \log p(\boldsymbol{\theta}^\star|\mathcal{D}) + \boldsymbol{\delta}^\top \mathbf{g} + \frac{1}{2}\boldsymbol{\delta}^\top \mathbf{H} \boldsymbol{\delta},$$

where the gradient and Hessian are, respectively,

$$\mathbf{g} = \nabla \log p(\boldsymbol{\theta}|\mathcal{D})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^\star} \qquad \mathbf{H} = \nabla\nabla \log p(\boldsymbol{\theta}|\mathcal{D})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^\star}.$$

Laplace approximation, in general

In general: suppose we have found the value $\boldsymbol{\theta}^\star$ that maximizes of $\log p(\boldsymbol{\theta}|\mathcal{D})$.

- Take a Taylor expansion of $\log p(\boldsymbol{\theta}|\mathcal{D})$ in the vicinity of $\boldsymbol{\theta}$. For small $\boldsymbol{\delta}$,

$$\log p(\boldsymbol{\theta}^\star + \boldsymbol{\delta}|\mathcal{D}) \approx \log p(\boldsymbol{\theta}^\star|\mathcal{D}) + \boldsymbol{\delta}^\top \mathbf{g} + \frac{1}{2}\boldsymbol{\delta}^\top \mathbf{H} \boldsymbol{\delta},$$

  where the gradient and Hessian are, respectively,

$$\mathbf{g} = \nabla \log p(\boldsymbol{\theta}|\mathcal{D})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^\star} \qquad \mathbf{H} = \nabla\nabla \log p(\boldsymbol{\theta}|\mathcal{D})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^\star}.$$

- Since $\boldsymbol{\theta}^\star$ is a mode of $\log p(\boldsymbol{\theta}|\mathcal{D})$, $\mathbf{g} = 0$, so taking $\boldsymbol{\theta} = \boldsymbol{\theta}^\star + \boldsymbol{\delta}$ we have

$$\log p(\boldsymbol{\theta}|\mathcal{D}) \approx \log p(\boldsymbol{\theta}^\star|\mathcal{D}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star)^\top \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}^\star).$$

Laplace approximation, in general

Now note that the first term is constant w.r.t. $\boldsymbol{\theta}$,

$$\log p(\boldsymbol{\theta}|\mathcal{D}) \approx \underbrace{\log p(\boldsymbol{\theta}^\star|\mathcal{D})}_{\text{const}} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star)^\top \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star).$$

Laplace approximation, in general

Now note that the first term is constant w.r.t. $\boldsymbol{\theta}$,

$$\log p(\boldsymbol{\theta}|\mathcal{D}) \approx \underbrace{\log p(\boldsymbol{\theta}^\star|\mathcal{D})}_{\text{const}} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star)^\top \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star).$$

This suggests a Gaussian approximation with $\boldsymbol{\Lambda} = -\mathbf{H} = -\nabla\nabla \log p(\boldsymbol{\theta}|\mathcal{D})$,

$$\log p(\boldsymbol{\theta}|\mathcal{D}) \approx \log \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star, \boldsymbol{\Lambda}^{-1}) = -\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star)^\top \boldsymbol{\Lambda}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star) + \text{const}.$$

Laplace approximation, in general

Now note that the first term is constant w.r.t. $\boldsymbol{\theta}$,

$$\log p(\boldsymbol{\theta}|\mathcal{D}) \approx \underbrace{\log p(\boldsymbol{\theta}^\star|\mathcal{D})}_{\text{const}} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star)^\top \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star).$$

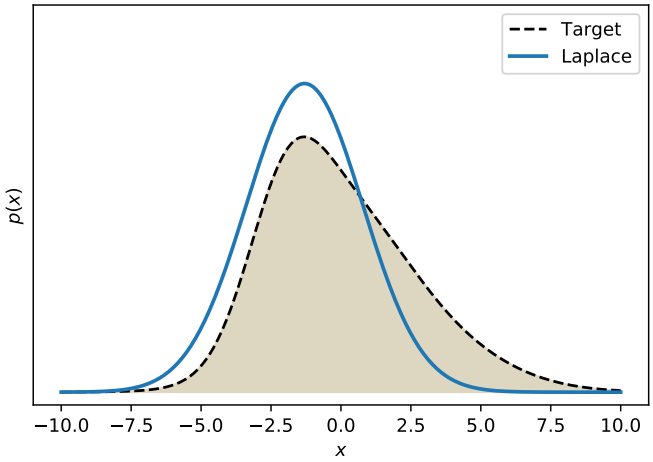
This suggests a Gaussian approximation with $\boldsymbol{\Lambda} = -\mathbf{H} = -\nabla\nabla \log p(\boldsymbol{\theta}|\mathcal{D})$,

$$\log p(\boldsymbol{\theta}|\mathcal{D}) \approx \log \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star, \boldsymbol{\Lambda}^{-1}) = -\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star)^\top \boldsymbol{\Lambda}(\boldsymbol{\theta} - \boldsymbol{\theta}^\star) + \text{const}.$$

Two apparent problems:

- Neural network posteriors are not even remotely Gaussian
- The Hessian for large networks will be impossible to compute or store

Laplace approximation cartoon



Laplace approximations in deep learning

Model (for regression problems):

$$p(y_i|\mathbf{x}_i, \boldsymbol{\theta}, \beta) = \mathcal{N}(y_i|f_{\boldsymbol{\theta}}(\mathbf{x}_i), \beta^{-1})$$
$$p(\boldsymbol{\theta}|\alpha) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

Here, $f_{\boldsymbol{\theta}}(\mathbf{x})$ is the neural network model, and $\alpha, \beta > 0$ are precisions (inverse variances).

MacKay (1991, 1992); presentation here follows Bishop PRML Ch.5.

Laplace approximations in deep learning

Model (for regression problems):

$$p(y_i|\mathbf{x}_i, \boldsymbol{\theta}, \beta) = \mathcal{N}(y_i|f_{\boldsymbol{\theta}}(\mathbf{x}_i), \beta^{-1})$$
$$p(\boldsymbol{\theta}|\alpha) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

Here, $f_{\boldsymbol{\theta}}(\mathbf{x})$ is the neural network model, and $\alpha, \beta > 0$ are precisions (inverse variances).

The overall posterior distribution (up to a constant) is given by

$$p(\boldsymbol{\theta}|\mathcal{D}, \alpha, \beta) = p(\boldsymbol{\theta}|\alpha) \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta}, \beta).$$

MacKay (1991, 1992); presentation here follows Bishop PRML Ch.5.

Laplace approximation: Step 1

The first step in fitting a Laplace approximation is to **find a posterior mode**.

The good news, is that this is exactly what "normal" training of deep learning models does! Maximize the log likelihood

$$\begin{aligned}\log p(\boldsymbol{\theta}|\mathcal{D}, \alpha, \beta) &= \log p(\boldsymbol{\theta}|\alpha) + \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}, \beta) \\ &= -\frac{\alpha}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta} - \frac{\beta}{2}\sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2 + \text{const.}\end{aligned}$$

to find the **MAP** estimate $\boldsymbol{\theta}^\star$.

Laplace approximation: Step 2

The second step is to compute the Hessian. The precision matrix in the Laplace approximation is

$$\mathbf{\Lambda} = -\nabla\nabla \log p(\boldsymbol{\theta}|\mathcal{D}, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H}$$

where $\mathbf{H}$ is the Hessian of the sum of squares

$$\begin{aligned}\mathbf{H} &= \nabla\nabla \sum_{i=1}^{N} (f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2 \\ &= \sum_{i=1}^{N} \nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)(\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i))^\top + \sum_{i=1}^{N} (f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)\nabla\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i).\end{aligned}$$

Computing $\nabla\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)$ can be **slow**.

Laplace approximation: Step 2

There are many ways to deal with the Hessian if computing the full matrix is infeasible (all the way down to a diagonal approximation).

One common choice is to approximate it with the outer product of the gradients:

$$\mathbf{H} = \sum_{i=1}^{N} \nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)(\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i))^\top + \sum_{i=1}^{N} (f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)\nabla\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

Laplace approximation: Step 2

There are many ways to deal with the Hessian if computing the full matrix is infeasible (all the way down to a diagonal approximation).

One common choice is to approximate it with the outer product of the gradients:

$$\begin{aligned}\mathbf{H} &= \sum_{i=1}^{N} \nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)(\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i))^\top + \sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)\nabla\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i) \\ &\approx \sum_{i=1}^{N} \nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)(\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i))^\top.\end{aligned}$$

Laplace approximation: Step 2

There are many ways to deal with the Hessian if computing the full matrix is infeasible (all the way down to a diagonal approximation).

One common choice is to approximate it with the outer product of the gradients:

$$\begin{aligned}\mathbf{H} &= \sum_{i=1}^{N} \nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)(\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i))^\top + \sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)\nabla\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i) \\ &\approx \sum_{i=1}^{N} \nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)(\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i))^\top.\end{aligned}$$

The intuition here is that the residual $(f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)$ in the second term should be small at the mode $\boldsymbol{\theta}^\star$, and in particular it hopefully should be in expectation zero (and independent of $\nabla\nabla f_{\boldsymbol{\theta}}(\mathbf{x}_i)$).

Laplace approximation: Step 3

Once we have the matrix $\mathbf{H}$, or some usable approximation, then with $\mathbf{\Lambda} = \alpha\mathbf{I} + \beta\mathbf{H}$ we have

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx q(\boldsymbol{\theta}|\mathcal{D}) \equiv \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star, \mathbf{\Lambda}^{-1}).$$

Laplace approximation: Step 3

Once we have the matrix $\mathbf{H}$, or some usable approximation, then with $\mathbf{\Lambda} = \alpha\mathbf{I} + \beta\mathbf{H}$ we have

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx q(\boldsymbol{\theta}|\mathcal{D}) \equiv \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star, \mathbf{\Lambda}^{-1}).$$

Now, how do we make predictions?

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \boldsymbol{\theta}) q(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}$$

is still intractable, since

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|f_{\boldsymbol{\theta}}(\mathbf{x}), \beta^{-1})$$

contains the non-linear $f_{\boldsymbol{\theta}}(\mathbf{x})$.

Making predictions by linearization

A "modern" approach might be to just draw many Monte Carlo samples from $q(\boldsymbol{\theta}|\mathcal{D})$ and evaluate the network, with

$$p(y|\mathbf{x}, \mathcal{D}) \approx \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(y|f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}), \beta^{-1}), \qquad \boldsymbol{\theta}^{(k)} \sim q(\boldsymbol{\theta}|\mathcal{D}).$$

Making predictions by linearization

A "modern" approach might be to just draw many Monte Carlo samples from $q(\boldsymbol{\theta}|\mathcal{D})$ and evaluate the network, with

$$p(y|\mathbf{x}, \mathcal{D}) \approx \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(y|f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}), \beta^{-1}), \qquad \boldsymbol{\theta}^{(k)} \sim q(\boldsymbol{\theta}|\mathcal{D}).$$

However, an alternative is to approximate the network with a first-order Taylor expansion, which can then be integrated exactly:

$$f_{\boldsymbol{\theta}}(\mathbf{x}) \approx f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}^\star), \text{ where}$$
$$\mathbf{g}(\mathbf{x}) = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^\star}$$

Making predictions by linearization

A "modern" approach might be to just draw many Monte Carlo samples from $q(\boldsymbol{\theta}|\mathcal{D})$ and evaluate the network, with

$$p(y|\mathbf{x}, \mathcal{D}) \approx \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(y|f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}), \beta^{-1}), \qquad \boldsymbol{\theta}^{(k)} \sim q(\boldsymbol{\theta}|\mathcal{D}).$$

However, an alternative is to approximate the network with a first-order Taylor expansion, which can then be integrated exactly:

$$f_{\boldsymbol{\theta}}(\mathbf{x}) \approx f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}^\star), \text{ where}$$
$$\mathbf{g}(\mathbf{x}) = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^\star}$$

so

$$p(y|\mathbf{x}, \boldsymbol{\theta}, \beta) \approx \mathcal{N}(y|f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}^\star), \beta^{-1}).$$

Making predictions by linearization

We can use this to approximate the predictive distribution as

$$p(y|\mathbf{x}, \mathcal{D}) \approx \int \mathcal{N}(y|f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^\star), \beta^{-1})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star, \boldsymbol{\Lambda}^{-1})d\boldsymbol{\theta}.$$

Making predictions by linearization

We can use this to approximate the predictive distribution as

$$p(y|\mathbf{x}, \mathcal{D}) \approx \int \mathcal{N}(y|f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^\star), \beta^{-1})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star, \boldsymbol{\Lambda}^{-1})d\boldsymbol{\theta}.$$

A bit of linear-Gaussian math yields an overall approximate predictive distribution

$$p(y|\mathbf{x}, \mathcal{D}, \alpha, \beta) \approx \mathcal{N}(y|f_{\boldsymbol{\theta}^\star}(\mathbf{x}), \sigma^2(\mathbf{x})),$$

whose mean is the MAP estimate and with variance that depends on the input value as

$$\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}(\mathbf{x})^\top \boldsymbol{\Lambda}^{-1}\mathbf{g}(\mathbf{x}).$$

Making predictions by linearization

We can use this to approximate the predictive distribution as

$$p(y|\mathbf{x}, \mathcal{D}) \approx \int \mathcal{N}(y|f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^\star), \beta^{-1})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star, \boldsymbol{\Lambda}^{-1})d\boldsymbol{\theta}.$$

A bit of linear-Gaussian math yields an overall approximate predictive distribution

$$p(y|\mathbf{x}, \mathcal{D}, \alpha, \beta) \approx \mathcal{N}(y|f_{\boldsymbol{\theta}^\star}(\mathbf{x}), \sigma^2(\mathbf{x})),$$

whose mean is the MAP estimate and with variance that depends on the input value as

$$\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}(\mathbf{x})^\top \boldsymbol{\Lambda}^{-1}\mathbf{g}(\mathbf{x}).$$

Worth taking a moment to look at these expressions!

Classification

That was regression, but classification is similar:

$$p(y = 1|\mathbf{x}, \mathcal{D}) \approx \int \sigma\left(f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^\star)\right) \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star, \boldsymbol{\Lambda}^{-1}) d\boldsymbol{\theta}.$$

- We can use a similar outer-product approximation of the Hessian; the only difference is that the likelihood changes
- The details for a logistic output are in Bishop PRML (and look a lot like the Hessian in your coursework), **but** it's not necessary to know this! You can actually just use automatic differentiation on the likelihood function, for any likelihood.
- For predictions, we can combine the "linearize the predictor" trick from this lecture with the "avoid high-dimensional Monte Carlo sampling" trick from the linear model example

Locally-linear classification

Combining "locally-linear" and "low-dimensional-integration" tricks:

$$p(y = 1|\mathbf{x}, \mathcal{D}) \approx \int \sigma\left(f_{\boldsymbol{\theta}}(\mathbf{x})\right) \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^{\star}, \boldsymbol{\Lambda}^{-1}) \mathrm{d}\boldsymbol{\theta}.$$

Locally-linear classification

Combining "locally-linear" and "low-dimensional-integration" tricks:

$$p(y=1|\mathbf{x},\mathcal{D}) \approx \int \sigma\left(f_{\boldsymbol{\theta}}(\mathbf{x})\right)\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star,\boldsymbol{\Lambda}^{-1})\mathrm{d}\boldsymbol{\theta}.$$

$$\approx \int \sigma\left(f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top(\boldsymbol{\theta}-\boldsymbol{\theta}^\star)\right)\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star,\boldsymbol{\Lambda}^{-1})\mathrm{d}\boldsymbol{\theta}.$$

Locally-linear classification

Combining "locally-linear" and "low-dimensional-integration" tricks:

$$
\begin{aligned}
p(y=1|\mathbf{x},\mathcal{D}) &\approx \int \sigma\left(f_{\boldsymbol{\theta}}(\mathbf{x})\right)\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star,\boldsymbol{\Lambda}^{-1})\mathrm{d}\boldsymbol{\theta}. \\
&\approx \int \sigma\left(f_{\boldsymbol{\theta}^\star}(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top(\boldsymbol{\theta}-\boldsymbol{\theta}^\star)\right)\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star,\boldsymbol{\Lambda}^{-1})\mathrm{d}\boldsymbol{\theta}. \\
&= \int \sigma\left(a\right)\mathcal{N}(a|f_{\boldsymbol{\theta}^\star}(\mathbf{x}),\mathbf{g}(\mathbf{x})^\top\boldsymbol{\Lambda}^{-1}\mathbf{g}(\mathbf{x}))\mathrm{d}a.
\end{aligned}
$$

Locally-linear classification

Combining "locally-linear" and "low-dimensional-integration" tricks:

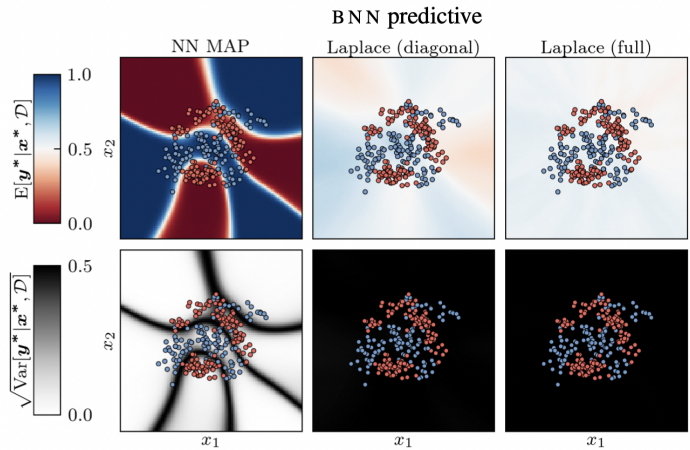
$$
\begin{aligned}
p(y=1|\mathbf{x},\mathcal{D}) &\approx \int \sigma\left(f_{\boldsymbol{\theta}}(\mathbf{x})\right)\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star,\boldsymbol{\Lambda}^{-1})\mathrm{d}\boldsymbol{\theta}. \\
&\approx \int \sigma\left(f_{\boldsymbol{\theta}^\star}(\mathbf{x})+\mathbf{g}(\mathbf{x})^\top(\boldsymbol{\theta}-\boldsymbol{\theta}^\star)\right)\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}^\star,\boldsymbol{\Lambda}^{-1})\mathrm{d}\boldsymbol{\theta}. \\
&= \int \sigma\left(a\right)\mathcal{N}(a|f_{\boldsymbol{\theta}^\star}(\mathbf{x}),\mathbf{g}(\mathbf{x})^\top\boldsymbol{\Lambda}^{-1}\mathbf{g}(\mathbf{x}))\mathrm{d}a.
\end{aligned}
$$

This can then be Monte Carlo integrated, or numerically approximated.

The weird thing...

It turns out you actually **need to linearize** the model in this way when making predictions!

Otherwise, there are **severe underfitting problems**.

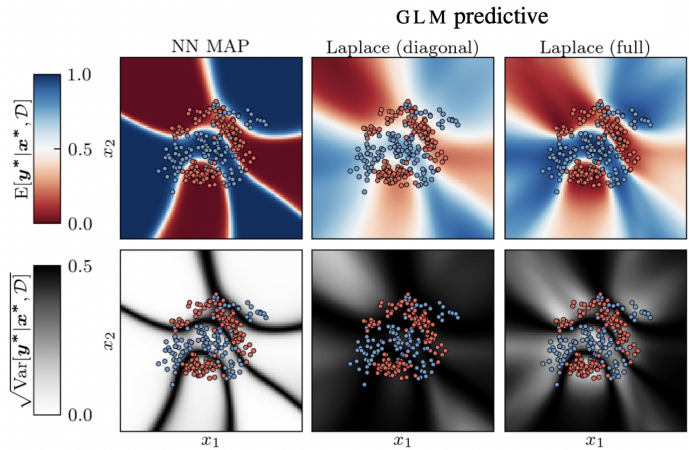


[Immer et al., 2020]

The weird thing...

It turns out you actually **need to linearize** the model in this way when making predictions!

Otherwise, there are **severe underfitting problems**.

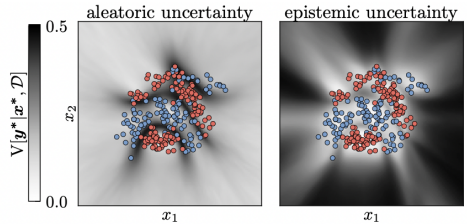


[Immer et al., 2020]

Uncertainty decomposition

It's been suggested that this might be necessary due to the quality of the Hessian approximations typically used.

Alternatively, it does make some sense that if we are doing inference in a simplified model (based on our Taylor expansion), then we should do prediction in that setting as well.



In any case, this also can provide us with reasonable uncertainty decomposition.

[Immer et al., 2020]

Summary: Laplace approximations

- Provides a Gaussian approximation to the posterior (and posterior predictive!) distribution
- Computing the Hessian gets expensive. Quite a bit of work on fast approximations:
  - Only compute on some of the weights (e.g. last layer only; or some other subset)
  - Block-diagonal approximations (Kronecker factorized)
- **Good**: The approximation has a tractable marginal likelihood, so it's possible to optimize hyperparameters α, β and do model comparison
- **Bad**: It only captures uncertainty around a single mode
- Recent summary / review paper: "Laplace Redux", Daxberger et al. (linked on Moodle)

Markov chain Monte Carlo

MCMC review

- It was a while ago that we talked about MCMC! Here's a quick recap:

MCMC review

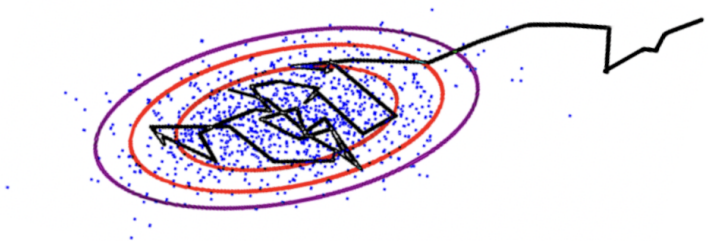
- It was a while ago that we talked about MCMC! Here's a quick recap:
  - Suppose we have a probability density $\pi(\boldsymbol{\theta})$ which we can evaluate (up to its normalizing constant), but cannot sample from
  - Starting with some initial point $\boldsymbol{\theta}^{(0)}$, define a Markov transition operator $A(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)})$, and use it to sample a long chain of values $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(T)}$.

MCMC review

- It was a while ago that we talked about MCMC! Here's a quick recap:
  - Suppose we have a probability density $\pi(\boldsymbol{\theta})$ which we can evaluate (up to its normalizing constant), but cannot sample from
  - Starting with some initial point $\boldsymbol{\theta}^{(0)}$, define a Markov transition operator $A(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)})$, and use it to sample a long chain of values $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(T)}$.
- **The tricky bit**: if you define $A(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)})$ just right, then after a while (e.g., for $t > T_0$) these values $\boldsymbol{\theta}^{(t)} \sim \pi(\boldsymbol{\theta})$.

MCMC cartoon

MCMC produces a sequence of correlated, approximate samples from a target distribution $\pi(\boldsymbol{\theta})$. It looks something like this:



This sequence follows a random walk, with Markov steps $A(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)})$.

Hopefully after the last coursework this looks somewhat familiar!

Figure: Iain Murray

Defining an MCMC transition

The way we defined MCMC operators before (and by far the most common approach) is to use the **Metropolis-Hastings** (MH) algorithm:

1. Given a value $\boldsymbol{\theta}^{(t)}$, sample a candidate value $\boldsymbol{\theta}'$ from a **proposal distribution** $q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{(t)})$
2. Compute the **acceptance ratio**

$$\alpha(\boldsymbol{\theta}', \boldsymbol{\theta}^{(t)}) = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}^{(t)})q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{(t)})}\right\}$$

3. With probability α, set $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}'$ (i.e., **accept** the proposal), otherwise **reject** and set $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$

Using MCMC samples

After running MCMC, we take K samples $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(K)}$ from our chain (e.g. by discarding initial values as "burnin").

These can then be used to compute Monte Carlo approximations to integrals. In particular, in a deep learning regression settings, we might have

$$p(y|\mathbf{x}, \mathcal{D}) \approx \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(y|f_{\boldsymbol{\theta}^{(k)}}(\mathbf{x}), \beta^{-1}).$$

Defining an MCMC transition

- The proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ is very flexible (few technical requirements), but a poor choice can make inference very slow
- For symmetric proposals (e.g. Gaussians centered at the current location), then the acceptance rate only depends on the ratio of $\pi(\boldsymbol{\theta}')$ and $\pi(\boldsymbol{\theta}^{(t)})$ and can be interpreted as "noisy" hill climbing,

$$\alpha(\boldsymbol{\theta}', \boldsymbol{\theta}^{(t)}) = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}^{(t)})}\right\}.$$

- As we saw in the coursework, getting the sampler parameters right can be "annoying" even for low-dimensional problems

MCMC challenges

Challenges:

- What makes a good proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ for high-dimensional $\boldsymbol{\theta}$?
- Will this algorithm, in practice, mix across different modes of the posterior?
- Can we scale this to large datasets?

Langevin dynamics: noisy gradient descent

Finding a single estimate of the weights is done by maximizing the log joint probability

$$\log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}).$$

This is done by taking gradient descent, with $\mathbf{g}_t = -\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}^{(t)}, \mathcal{D})$, and updating

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \mathbf{g}_t$$

for some small learning rate $\eta > 0$.

Langevin dynamics: noisy gradient descent

Finding a single estimate of the weights is done by maximizing the log joint probability

$$\log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}).$$

This is done by taking gradient descent, with $\mathbf{g}_t = -\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}^{(t)}, \mathcal{D})$, and updating

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \mathbf{g}_t$$

for some small learning rate $\eta > 0$. Langevin Monte Carlo includes an additional momentum variable $\mathbf{p} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and propose

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \frac{1}{2}\epsilon^2 \mathbf{g} + \epsilon \mathbf{p}.$$

Langevin dynamics: noisy gradient descent

Langevin Monte Carlo has a physical interpretation, and can be viewed as a special case of Hamiltonian Monte Carlo. If you're interested in this, see

- Radford Neal (2010). *MCMC using Hamiltonian dynamics*. In *Handbook of Markov Chain Monte Carlo*. `https://arxiv.org/abs/1206.1901`

Langevin dynamics: noisy gradient descent

Langevin Monte Carlo has a physical interpretation, and can be viewed as a special case of Hamiltonian Monte Carlo. If you're interested in this, see

- Radford Neal (2010). *MCMC using Hamiltonian dynamics*. In *Handbook of Markov Chain Monte Carlo*. `https://arxiv.org/abs/1206.1901`

For purposes of this lecture, we'll take a simplified view of the update

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \underbrace{\frac{1}{2}\epsilon^2}_{\eta} \mathbf{g} + \epsilon \underbrace{\mathbf{p}}_{\mathcal{N}(\mathbf{0},\mathbf{I})}$$

Langevin dynamics: noisy gradient descent

Langevin Monte Carlo has a physical interpretation, and can be viewed as a special case of Hamiltonian Monte Carlo. If you're interested in this, see

- Radford Neal (2010). *MCMC using Hamiltonian dynamics*. In *Handbook of Markov Chain Monte Carlo*. `https://arxiv.org/abs/1206.1901`

For purposes of this lecture, we'll take a simplified view of the update

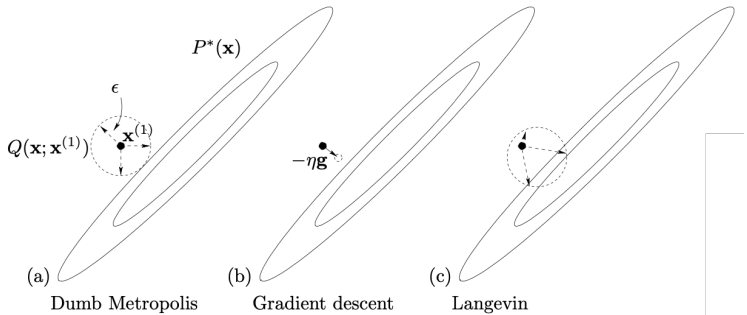
$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \underbrace{\frac{1}{2}\epsilon^2}_{\eta} \mathbf{g} + \epsilon \underbrace{\mathbf{p}}_{\mathcal{N}(\mathbf{0},\mathbf{I})},$$

which we can interpret as a Metropolis-Hastings proposal of the form

$$q(\boldsymbol{\theta}'|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}'|\boldsymbol{\theta} - \eta\mathbf{g}, \epsilon^2\mathbf{I}).$$

Langevin dynamics example

Including the gradient information into the proposal helps with performance in high-dimensional spaces, relative to e.g. simple Gaussian proposals.



(Hamiltonian Monte Carlo helps even more! It takes multiple gradient updates before the accept-reject step.)

Figure: MacKay (2003)

Dealing with big data?

A big challenge with these methods is that they require access to the entire dataset at every update:

$$\log \pi(\boldsymbol{\theta}) \equiv \log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

must be computed to evaluate the MH acceptance ratio, as well as for computing $\mathbf{g} = -\nabla \log \pi(\boldsymbol{\theta})$.

Dealing with big data?

A big challenge with these methods is that they require access to the entire dataset at every update:

$$\log \pi(\boldsymbol{\theta}) \equiv \log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

must be computed to evaluate the MH acceptance ratio, as well as for computing $\mathbf{g} = -\nabla \log \pi(\boldsymbol{\theta})$.

When finding point estimates in deep learning models, it's rare to compute gradients on the whole data at once! Instead, we use a **mini-batch** estimate

$$\log \hat{\pi}_M(\boldsymbol{\theta}) = \frac{N}{M} \sum_{j=1}^{M} \log p(y_j|\mathbf{x}_j, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

where $j = 1, \ldots, M$ are a random subset of the indices $i = 1, \ldots, N$, $M \ll N$.

Stochastic gradient Langevin dynamics

The mini-batch estimate $\hat{\pi}_M(\boldsymbol{\theta})$ with a subset of the data provides an unbiased estimate of the full-data $\pi(\boldsymbol{\theta})$.

- In optimization, we then take stochastic gradient steps

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \epsilon_t \hat{\mathbf{g}}_t$$

- As an MCMC proposal, we sample from the proposal distribution

$$q(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)}) = \mathcal{N}(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)} + \eta_t \hat{\mathbf{g}}_t, \epsilon_t^2 \mathbf{I})$$

where $\hat{\mathbf{g}}_t = \nabla \log \hat{\pi}_M(\boldsymbol{\theta}^{(t)})$.

Stochastic gradient Langevin dynamics

The mini-batch estimate $\hat{\pi}_M(\boldsymbol{\theta})$ with a subset of the data provides an unbiased estimate of the full-data $\pi(\boldsymbol{\theta})$.

- In optimization, we then take stochastic gradient steps

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \epsilon_t \hat{\mathbf{g}}_t$$

- As an MCMC proposal, we sample from the proposal distribution

$$q(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)}) = \mathcal{N}(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)} + \eta_t \hat{\mathbf{g}}_t, \epsilon_t^2 \mathbf{I})$$

where $\hat{\mathbf{g}}_t = \nabla \log \hat{\pi}_M(\boldsymbol{\theta}^{(t)})$.

This means that we can sample from the proposal distribution without evaluating all the data!

Stochastic gradient Langevin dynamics

One little problem: the Metropolis-Hastings acceptance ratio

$$\alpha(\boldsymbol{\theta}', \boldsymbol{\theta}^{(t)}) = \min \left\{ 1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}^{(t)})q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{(t)})} \right\}$$

still requires evaluating the target density.

Stochastic gradient Langevin dynamics

One little problem: the Metropolis-Hastings acceptance ratio

$$\alpha(\boldsymbol{\theta}', \boldsymbol{\theta}^{(t)}) = \min \left\{ 1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}^{(t)})q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{(t)})} \right\}$$

still requires evaluating the target density.

The solution of Stochastic gradient Langevin dynamics (SGLD; Welling & Teh 2011) is to **skip the accept/reject step entirely**!

Stochastic gradient Langevin dynamics

One little problem: the Metropolis-Hastings acceptance ratio

$$\alpha(\boldsymbol{\theta}', \boldsymbol{\theta}^{(t)}) = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}^{(t)})q(\boldsymbol{\theta}'|\boldsymbol{\theta}^{(t)})}\right\}$$

still requires evaluating the target density.

The solution of Stochastic gradient Langevin dynamics (SGLD; Welling & Teh 2011) is to **skip the accept/reject step entirely**!

This can be justified by observing that as $\epsilon \to 0$, the proposal is always accepted. For details: Neal (2010) and Welling & Teh (2011)

Stochastic gradient Langevin dynamics

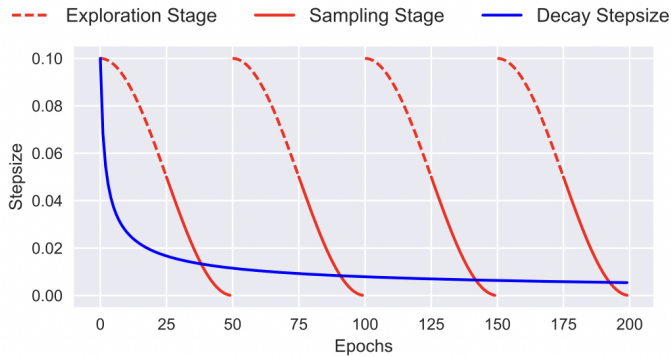
This means the overall algorithm takes the form

1. Sample
$$\mathbf{p}_t \sim \mathcal{N}(\mathbf{p}|\mathbf{0}, \mathbf{I})$$
2. Compute update
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \frac{1}{2}\epsilon_t^2 \nabla \log \hat{\pi}_M(\boldsymbol{\theta}^{(t)}) + \epsilon_t \mathbf{p}_t$$

For convergence to be guaranteed, SGLD needs to have a sequence of step sizes ϵ_t that decrease over time — this is only rarely done, though, due to computational cost.

Cyclic learning rates?

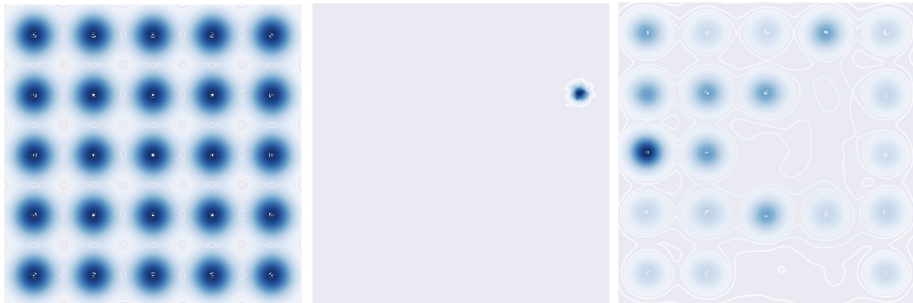
Mixing across modes is still hard! MCMC won't move between disconnected regions. One thing that can help is a cyclic learning rate, in which ϵ_t has a periodic behavior.



Need to be careful that "enough" steps are taken to actually find "real" samples!

[Zhang et al., 2019]

Mixing across modes



(a) Target (b) SGLD (c) cSGLD

Mixing across modes is easier with a cyclic learning rate schedule.

[Zhang et al., 2019]

Practical notes on stochastic MCMC

- It's worth noting that plain-old SGD (not SGLD!) with a constant learning rate can also be interpreted as an approximate sampler (Mandt et al., 2017). This is leveraged by methods like SWAG (Maddox et al., 2019)
- SGLD can be improved by including a preconditioning matrix (analogous to adaptive step-size methods in optimization)
- Stochastic Hamiltonian Monte Carlo can be picky about hyperparameters

Variational inference ("Bayes by backprop")

Variational inference

The Laplace approximation isn't the only way to find a Gaussian approximation to a neural network posterior.

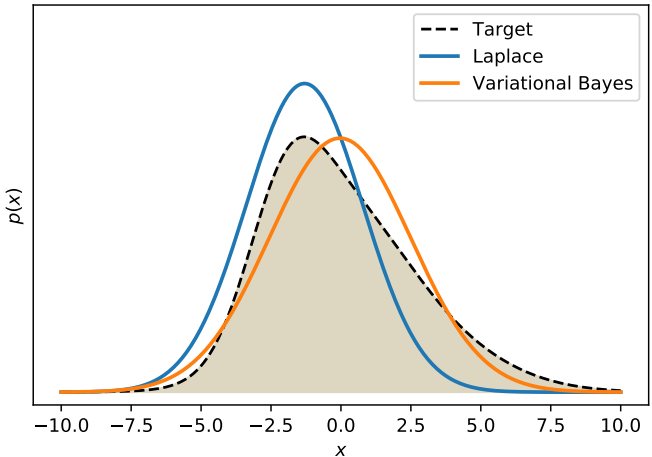
The other approach we've covered, **variational inference**, directly searches for a Gaussian

$$q_\lambda(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \approx p(\boldsymbol{\theta}|\mathcal{D}).$$

As with linear models, our approach is to minimize the KL divergence w.r.t. $\lambda = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$,

$$D_{KL}(q_\lambda(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}|\mathcal{D})) = \int q_\lambda(\boldsymbol{\theta}) \log \frac{q_\lambda(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathcal{D})} d\boldsymbol{\theta}.$$

Variational Bayes cartoon



Evidence lower bound (ELBO)

Nothing new yet: we will minimize the KL divergence by maximizing the **ELBO**

$$\underbrace{\mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right]}_{ELBO,\ \mathcal{L}(\lambda,\mathcal{D})} = \underbrace{\log p(\mathcal{D})}_{\text{const.}} - \underbrace{D_{KL}(q_\lambda(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}|\mathcal{D}))}_{\geq 0}$$

Evidence lower bound (ELBO)

Nothing new yet: we will minimize the KL divergence by maximizing the **ELBO**

$$\underbrace{\mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right]}_{ELBO,\ \mathcal{L}(\lambda,\mathcal{D})} = \underbrace{\log p(\mathcal{D})}_{\text{const.}} - \underbrace{D_{KL}(q_\lambda(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}|\mathcal{D}))}_{\geq 0}$$

$$\mathcal{L}(\lambda, \mathcal{D}) \leq \log p(\mathcal{D}).$$

Maximizing the ELBO by gradient ascent

Using reparameterized gradient estimators, we can compute

$$\nabla_\lambda \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_\lambda \log \frac{p(r(\lambda, \boldsymbol{\epsilon}), \mathcal{D})}{q_\lambda(r(\lambda, \boldsymbol{\epsilon}))}\right].$$

- In practice, draw a single sample $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$, and use it to compute $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$
- At a cost of $\times 2$ parameters, we also estimate a per-weight standard deviation (**bad**: diagonal only)
- Blundell et al. (2015), *Weight uncertainty in neural networks* writes out derivatives with respect to $\boldsymbol{\mu}, \boldsymbol{\sigma}$ explicitly. But knowing what we know now, we can just "solve" the VB inference problem using reparameterized sampling with autodiff.

Use mini-batch estimates of the gradient

For models where $\log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$, then we can happily use mini-batch estimators:

$$\mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right]$$

Use mini-batch estimates of the gradient

For models where $\log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$, then we can happily use mini-batch estimators:

$$\mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right] = \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right]$$

Use mini-batch estimates of the gradient

For models where $\log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$, then we can happily use mini-batch estimators:

$$\begin{aligned}
\mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right] &= \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})\right] + \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right]
\end{aligned}$$

Use mini-batch estimates of the gradient

For models where $\log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$, then we can happily use mini-batch estimators:

$$\begin{aligned}
\mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right] &= \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})\right] + \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right] \\
&= \sum_{i=1}^{N} \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})\right] - D_{KL}(q_\lambda(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}))
\end{aligned}$$

Use mini-batch estimates of the gradient

For models where $\log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$, then we can happily use mini-batch estimators:

$$
\begin{aligned}
\mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right] &= \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})\right] + \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right] \\
&= \sum_{i=1}^{N} \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})\right] - D_{KL}(q_\lambda(\boldsymbol{\theta}) \| p(\boldsymbol{\theta})) \\
&\approx \frac{N}{M} \sum_{j=1}^{M} \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log p(y_j|\mathbf{x}_j, \boldsymbol{\theta})\right] - D_{KL}(q_\lambda(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}))
\end{aligned}
$$

Use mini-batch estimates of the gradient

For models where $\log p(\boldsymbol{\theta}, \mathcal{D}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$, then we can happily use mini-batch estimators:

$$
\begin{aligned}
\mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q_\lambda(\boldsymbol{\theta})}\right] &= \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + \log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right] \\
&= \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})\right] + \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\right] \\
&= \sum_{i=1}^{N} \mathbb{E}_{q_\lambda(\boldsymbol{\theta})}\left[\log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})\right] - D_{KL}(q_\lambda(\boldsymbol{\theta})\|p(\boldsymbol{\theta})) \\
&\approx \frac{N}{M} \sum_{j=1}^{M} \mathbb{E}_{p(\epsilon)}\left[\log p(y_j|\mathbf{x}_j, r(\lambda, \boldsymbol{\epsilon}))\right] - D_{KL}(q_\lambda(\boldsymbol{\theta})\|p(\boldsymbol{\theta}))
\end{aligned}
$$

Reducing Monte Carlo integration

If the priors and posteriors on the weights are Gaussian, this can be made more efficient for linear layers by using the "local reparameterization trick", from Kingma et al. (2015):

- Suppose the entries of $\mathbf{A} \in \mathbb{R}^{H \times D}, \mathbf{b} \in \mathbb{R}^{H}$ are Gaussian distributed
- Then for any given input $\mathbf{x}$, the value $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{b} \in \mathbb{R}^{H}$ is also Gaussian distributed, but with a much lower dimensionality
- Sampling $\mathbf{h}$ directly only requires drawing H random values, as opposed to $(H + 1)D$, which reduces variance in the estimated loss and gradients

Some comments on "Bayes by backprop"

- It's unclear whether or not a factorized posterior (i.e. with diagonal matrix Σ) is problematic.

Some comments on "Bayes by backprop"

- It's unclear whether or not a factorized posterior (i.e. with diagonal matrix Σ) is problematic.
- Other priors and posteriors are possible, but not commonly used. (Occasional prior choices: Laplace distributions; scale mixtures of Gaussians. Posteriors could have low-rank-plus-diagonal $\Sigma \ldots$)

Some comments on "Bayes by backprop"

- It's unclear whether or not a factorized posterior (i.e. with diagonal matrix Σ) is problematic.
- Other priors and posteriors are possible, but not commonly used. (Occasional prior choices: Laplace distributions; scale mixtures of Gaussians. Posteriors could have low-rank-plus-diagonal $\Sigma \ldots$)
- This can be annoyingly hard to train. Often it's important to initialize the optimization process by first finding a MAP estimate for $\boldsymbol{\mu}$, and initializing the variances to be very small.

Ensembles and other approaches

In practice, what should we do?

Often when people want to quickly get an uncertainty estimate out of their deep learning model, they necessarily don't do any of the things we've looked at so far.

In practice, what should we do?

Often when people want to quickly get an uncertainty estimate out of their deep learning model, they necessarily don't do any of the things we've looked at so far.

Three common choices:

- Ensembles of models, each trained from different initializations (Lakshminarayanan et al. 2017, and others)

In practice, what should we do?

Often when people want to quickly get an uncertainty estimate out of their deep learning model, they necessarily don't do any of the things we've looked at so far.

Three common choices:

- Ensembles of models, each trained from different initializations (Lakshminarayanan et al. 2017, and others)
- Monte Carlo Dropout (Gal & Ghahramani, 2016)

In practice, what should we do?

Often when people want to quickly get an uncertainty estimate out of their deep learning model, they necessarily don't do any of the things we've looked at so far.

Three common choices:

- Ensembles of models, each trained from different initializations (Lakshminarayanan et al. 2017, and others)
- Monte Carlo Dropout (Gal & Ghahramani, 2016)
- Deterministic networks, with a Bayesian linear model as the final layer (could be "exact" for regression, or using VB or Laplace approximations for classification)
  - Ideally this would be fit jointly, rather than pre-training the network and keeping the "features" fixed...

In practice, what should we do?

Often when people want to quickly get an uncertainty estimate out of their deep learning model, they necessarily don't do any of the things we've looked at so far.

Three common choices:

- Ensembles of models, each trained from different initializations (Lakshminarayanan et al. 2017, and others)
- Monte Carlo Dropout (Gal & Ghahramani, 2016)
- Deterministic networks, with a Bayesian linear model as the final layer (could be "exact" for regression, or using VB or Laplace approximations for classification)
  - Ideally this would be fit jointly, rather than pre-training the network and keeping the "features" fixed...

You can also get uncertainty estimates from unlikely sources — for example, using data augmentation at test time!

You can also take ensembles of other methods!

For example, taking ensembles of methods that produce local Gaussian approximations would yield a mixture-of-Gaussians posterior.

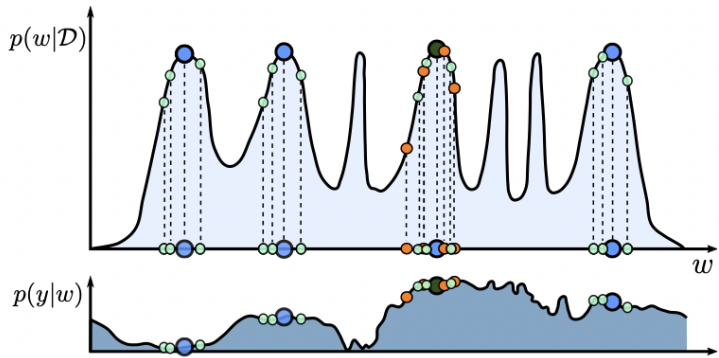


Figure: Wilson & Izmailov, 2020

Summary

- Local Gaussian approximations can be useful for deep learning models, despite multimodality in the posterior
- Stochastic gradient MCMC methods are more able to mix across modes and can scale, but they are sensitive to parameter tuning issues
- Challenges for Laplace approximations are largely computational
- Challenges for VB lie in optimization, as well as opening up conceptual questions (what sort of posterior approximation to use, etc)