Probabilistic Programming for Bayesian Computation: Part I

Harsha Veeramachaneni

2018/01/01

Probabilistic Models of Physical Systems

- Loosely speaking, a probabilistic model of a physical process is an origin story of observations from that process
- The probabilistic model encodes our assumptions about how to generate data that *resembles* observations from that system
- More formally it specifies how to sample observations from the joint probability distribution

$$P(\mathbf{x}, \mathbf{z}|\theta) = P(\mathbf{x}|\mathbf{z}, \theta)P(\mathbf{z}|\theta)$$

  - where θ are parameters of the model, $\mathbf{x}$ are the observations and $\mathbf{z}$ are unobserved or **latent** variables
  - think of $\mathbf{z}$ as the observations we wish we had

Probabilistic Models of Physical Systems

- Loosely speaking, a probabilistic model of a physical process is an origin story of observations from that process
- The probabilistic model encodes our assumptions about how to generate data that *resembles* observations from that system
- More formally it specifies how to sample observations from the joint probability distribution

$$P(\mathbf{x}, \mathbf{z}|\theta) = P(\mathbf{x}|\mathbf{z}, \theta)P(\mathbf{z}|\theta)$$

  - where θ are parameters of the model, $\mathbf{x}$ are the observations and $\mathbf{z}$ are unobserved or **latent** variables
  - think of $\mathbf{z}$ as the observations we wish we had

The distinction between $\mathbf{z}$ and θ is somewhat arbitrary. Usually θ are parameters that are known or are to be optimized over, and $\mathbf{z}$ are latent variables whose distributions we wish to infer

Probabilistic Model Desiderata

- We would like to have models that enable
  - **Simulation** of the physical system, i.e., generate samples

$$\mathbf{x}, \mathbf{z} \sim P(\mathbf{x}, \mathbf{z}|\theta)$$

Probabilistic Model Desiderata

- We would like to have models that enable
  - **Simulation** of the physical system, i.e., generate samples

    $$\mathbf{x}, \mathbf{z} \sim P(\mathbf{x}, \mathbf{z}|\theta)$$

  - **Optimization**, driving the observations to desired values, e.g.,

    $$\arg\max_\theta E[g(\mathbf{x})|\theta]$$

    (maximum likelihood estimate of θ given data $\mathbf{x}_o$, can be obtained by setting $g(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_o)$)

Probabilistic Model Desiderata

- We would like to have models that enable
  - **Simulation** of the physical system, i.e., generate samples

    $$\mathbf{x}, \mathbf{z} \sim P(\mathbf{x}, \mathbf{z}|\theta)$$

  - **Optimization**, driving the observations to desired values, e.g.,

    $$\arg\max_\theta E[g(\mathbf{x})|\theta]$$

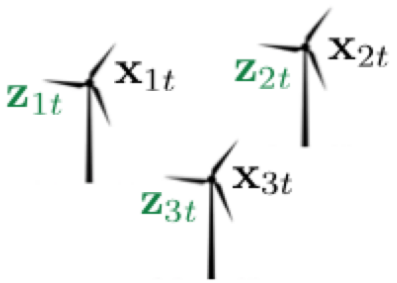
    (maximum likelihood estimate of θ given data $\mathbf{x}_o$, can be obtained by setting $g(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_o)$)
  - **Inference**, i.e., answer conditional queries such as

    $$\mathrm{E}[f(\mathbf{z})|\mathbf{x} = \mathbf{x}_o, \theta]$$

Example: Wind Farm Wake Estimation

- We might have a parameterized wake model that models waked wind-speeds from the geometry of the turbines, their thrust coefficients, and the free-stream wind-speeds
- We may wish to fit the wake model to the observed wind-speeds and estimate the free-stream wind-speeds

$$\text{Unobserved free-stream wind-speeds at time } t : \mathbf{z}_t \triangleq [\mathbf{z}_{1t}, \mathbf{z}_{2t}, \mathbf{z}_{3t}]$$
$$\text{Observed waked wind-speeds at time } t : \mathbf{x}_t \triangleq [\mathbf{x}_{1t}, \mathbf{x}_{2t}, \mathbf{x}_{3t}]$$
$$\text{Paramerized wake model} : f(.\,, \lambda)$$



$$\mathbf{z}_{it} \sim \text{Weibull}(\alpha_i, \beta_i)$$
$$\mathbf{x}_t \sim \text{Normal}(f(\mathbf{z}_t, \lambda), \epsilon I)$$

Example: Load Disaggregtion

- We are given the **sum** of m different time-series each of which is generated from a known (and different) Markov process
- We wish to infer all the individual constituent time-series from their sum
- Usually comes up in the disaggregation of the electrical consumption of individual devices at a building from the aggregate consumption time-series at the meter

$$\text{Load time-series from load } i : \mathbf{z}_i \triangleq [\mathbf{z}_{11}, \ldots, \mathbf{z}_{1T}]$$
$$\text{Observed aggregate load} : \mathbf{x} \triangleq [\mathbf{x}_1, \ldots, \mathbf{x}_T]$$

$$\mathbf{z}_i \sim P_i(\theta_i)$$
$$\mathbf{x} \sim \text{Normal}(\sum_i^m \mathbf{z}_i, \epsilon I)$$

Probabilistic Programming Languages

- Probabilistic programming languages allow the declaration of probabilistic models by composing deterministic and stochastic computation
- Their goal is to separate the concerns of *model specification* and *model inference* (the former requires *domain* expertise, the latter *statistical* expertise)
- Either extend existing languages (PyMC3, PyRo, Turing.jl etc.) or have their own self-contained syntax (Stan, BUGS etc.)
- Identified as a foundational technology for future machine learning systems by government funding agencies (DARPA), and large corporations (Google, Microsoft, etc.) are heavily investing in this area

Nice Probabilistic Models

- We assume that
  - We can sample from the model

$$\mathbf{x}, \mathbf{z} \sim P(\mathbf{x}, \mathbf{z}|\theta)$$

    i.e., simulation is built in

Nice Probabilistic Models

- We assume that
  - We can sample from the model

    $$\mathbf{x}, \mathbf{z} \sim P(\mathbf{x}, \mathbf{z}|\theta)$$

    i.e., simulation is built in
  - We can calculate forward joint probability of all variables, aka likelihood,

    $$P(\mathbf{x}, \mathbf{z}|\theta)$$

Nice Probabilistic Models

- We assume that
  - We can sample from the model

    $$\mathbf{x}, \mathbf{z} \sim P(\mathbf{x}, \mathbf{z}|\theta)$$

    i.e., simulation is built in
  - We can calculate forward joint probability of all variables, aka likelihood,

    $$P(\mathbf{x}, \mathbf{z}|\theta)$$

  - The forward probabilities are differentiable with respect to their parameters and the gradients $\nabla_\theta P(\mathbf{x}|\mathbf{z}, \theta)$ and $\nabla_\theta P(\mathbf{z}|\theta)$ are *easy* to compute

Nice Probabilistic Models

- We assume that
  - We can sample from the model

    $$\mathbf{x}, \mathbf{z} \sim P(\mathbf{x}, \mathbf{z}|\theta)$$

    i.e., simulation is built in
  - We can calculate forward joint probability of all variables, aka likelihood,

    $$P(\mathbf{x}, \mathbf{z}|\theta)$$

  - The forward probabilities are differentiable with respect to their parameters and the gradients $\nabla_\theta P(\mathbf{x}|\mathbf{z}, \theta)$ and $\nabla_\theta P(\mathbf{z}|\theta)$ are *easy* to compute
- The rest of the talk is about **inference** for such *nice* models

Nice Probabilistic Models

- We assume that
  - We can sample from the model

    $$\mathbf{x}, \mathbf{z} \sim P(\mathbf{x}, \mathbf{z}|\theta)$$

    i.e., simulation is built in
  - We can calculate forward joint probability of all variables, aka likelihood,

    $$P(\mathbf{x}, \mathbf{z}|\theta)$$

  - The forward probabilities are differentiable with respect to their parameters and the gradients $\nabla_\theta P(\mathbf{x}|\mathbf{z}, \theta)$ and $\nabla_\theta P(\mathbf{z}|\theta)$ are *easy* to compute
- The rest of the talk is about **inference** for such *nice* models
- But before that, a quick aside ...

Score Function Gradient Estimator or the REINFORCE trick

- Assume we want to perform the optimization $\arg\max_\theta E[g(\mathbf{x})|\theta]$ for some function g
- We need the gradient $\nabla_\theta \mathrm{E}[g(\mathbf{x})|\theta]$

Score Function Gradient Estimator or the REINFORCE trick

- Assume we want to perform the optimization $\arg\max_\theta E[g(\mathbf{x})|\theta]$ for some function g
- We need the gradient $\nabla_\theta \mathrm{E}[g(\mathbf{x})|\theta]$

$$
\begin{aligned}
\nabla_\theta \mathrm{E}[g(\mathbf{x})|\theta] &= \nabla_\theta \int g(\mathbf{x}) P(\mathbf{x}|\theta) d\mathbf{x} \\
&= \int g(\mathbf{x}) \nabla_\theta P(\mathbf{x}|\theta) d\mathbf{x} \\
&= \int g(\mathbf{x}) P(\mathbf{x}|\theta) \nabla_\theta \log(P(\mathbf{x}|\theta)) d\mathbf{x} \\
&= \mathrm{E}[g(\mathbf{x}) \nabla_\theta \log(P(\mathbf{x}|\theta))|\theta] \\
&\approx \frac{1}{K} \sum_i^K g(\mathbf{x}_i) \nabla_\theta \log(P(\mathbf{x}_i|\theta))]
\end{aligned}
$$

Score Function Gradient Estimator or the REINFORCE trick

- Assume we want to perform the optimization $\arg\max_\theta E[g(\mathbf{x})|\theta]$ for some function g
- We need the gradient $\nabla_\theta \mathrm{E}[g(\mathbf{x})|\theta]$

$$
\begin{aligned}
\nabla_\theta \mathrm{E}[g(\mathbf{x})|\theta] &= \nabla_\theta \int g(\mathbf{x}) P(\mathbf{x}|\theta) d\mathbf{x} \\
&= \int g(\mathbf{x}) \nabla_\theta P(\mathbf{x}|\theta) d\mathbf{x} \\
&= \int g(\mathbf{x}) P(\mathbf{x}|\theta) \nabla_\theta \log(P(\mathbf{x}|\theta)) d\mathbf{x} \\
&= \mathrm{E}[g(\mathbf{x}) \nabla_\theta \log(P(\mathbf{x}|\theta))|\theta] \\
&\approx \frac{1}{K} \sum_i^K g(\mathbf{x}_i) \nabla_\theta \log(P(\mathbf{x}_i|\theta))]
\end{aligned}
$$

- We can estimate the gradient of the expectation by averaging some function calculated on samples
  - is unbiased but generally has high variance
- We'll use this trick for inference

Example of Modeling in a PPL

- Assume we have 4 independent readings from a noisy Celcius thermometer and 20 independent readings from noisy Farenheit thermometer (c_i and f_i respectively) of a liquid at constant temperature t
- Assume further that we know that the noise is Gaussian with std. devs. $\sigma_c = 2$ and $\sigma_f = 6$, for the Celcius and Farenheit thermometers resp.
- We want to infer the true temperature t in Celcius
- In this example

$$\theta \triangleq (\sigma_c, \sigma_f, \ldots)$$
$$\mathbf{z} \triangleq t$$
$$\mathbf{x} \triangleq (c_1, c_2, \ldots, c_{10}, f_1, \ldots, f_4)$$

Example of Modeling in a PPL

- Assume we have 4 independent readings from a noisy Celcius thermometer and 20 independent readings from noisy Farenheit thermometer (c_i and f_i respectively) of a liquid at constant temperature t
- Assume further that we know that the noise is Gaussian with std. devs. $\sigma_c = 2$ and $\sigma_f = 6$, for the Celcius and Farenheit thermometers resp.
- We want to infer the true temperature t in Celcius
- In this example

$$\begin{aligned}\theta &\triangleq (\sigma_c, \sigma_f, \ldots)\\ \mathbf{z} &\triangleq t\\ \mathbf{x} &\triangleq (c_1, c_2, \ldots, c_{10}, f_1, \ldots, f_4)\end{aligned}$$

- What if we a priori believe that t is distributed as $t \sim \mathrm{Normal}(20, 5)$

Example of Modeling in a PPL

- Assume we have 4 independent readings from a noisy Celcius thermometer and 20 independent readings from noisy Farenheit thermometer (c_i and f_i respectively) of a liquid at constant temperature t
- Assume further that we know that the noise is Gaussian with std. devs. $\sigma_c = 2$ and $\sigma_f = 6$, for the Celcius and Farenheit thermometers resp.
- We want to infer the true temperature t in Celcius
- In this example

$$\begin{aligned}\theta &\triangleq (\sigma_c, \sigma_f, \ldots)\\ \mathbf{z} &\triangleq t\\ \mathbf{x} &\triangleq (c_1, c_2, \ldots, c_{10}, f_1, \ldots, f_4)\end{aligned}$$

- What if we a priori believe that t is distributed as $t \sim \mathrm{Normal}(20, 5)$
- What if instead we only know that t has to be in the range $t \in [15, 50]$

Programming in Stan

- Program the model

```
temperature_model <- "
data {
  int<lower=0> Nf; // number of faren samples
  int<lower=0> Nc; // number of celcius samples
  real farenheit_sigma; //error of the farenheit thermometer
  real celcius_sigma;  //error of the celcius thermometer
  real farenheit_reported[Nf];  //data from the farenheit thermometer
  real celcius_reported[Nc]; //data from the celcius thermometer
}

parameters {
  real<lower=15,upper=50> true_temp;
}

model {
  farenheit_reported ~ normal(true_temp * 1.8 + 32, farenheit_sigma);
  celcius_reported ~ normal(true_temp, celcius_sigma);
}"
```

Bayesian Inference

- What does the observed data tell us about unobserved variables in our model?

$$\begin{aligned}\mathrm{E}[f(\mathbf{z})|\mathbf{x}=\mathbf{x}_o] &= \int f(\mathbf{z})P(\mathbf{z}|\mathbf{x}=\mathbf{x}_o)d\mathbf{z} \\ &= \int f(\mathbf{z})\frac{P(\mathbf{x}=\mathbf{x}_o|\mathbf{z})P(\mathbf{z})}{P(\mathbf{x}=\mathbf{x}_o)}d\mathbf{z}\end{aligned}$$

- For all but the simplest models, the *posterior* distribution $P(\mathbf{z}|\mathbf{x}=\mathbf{x}_o)$ is difficult to compute or sample from, let alone integrate over
- The difficult part is the computation of the marginal probability $P(\mathbf{x}=\mathbf{x}_o)=\int P(\mathbf{x}=\mathbf{x}_o|\mathbf{z})P(\mathbf{z})d\mathbf{z}$
  - $\log(P(\mathbf{x}=\mathbf{x}_o))$ is called the *evidence*
- Most of the research in Bayesian inference is about estimating the above integral for general models

Back to Example in Stan

- Generate some data

```
true_temp <- 20  #unknown value
celcius_sigma <- 2 #known
farenheit_sigma <- 6 #known
Nf <- 20
Nc <- 4

farenheit_reported <-  rnorm(Nf, true_temp * 1.8 + 32, sd=farenheit_sigma)
celcius_reported <-  rnorm(Nc, true_temp, sd=celcius_sigma)
```

- Perform Inference

```
fit <- stan(model_code = temperature_model, pars = c("true_temp"),
            data = list(Nf = Nf, Nc = Nc,
                        farenheit_sigma = farenheit_sigma,
                        celcius_sigma = celcius_sigma,
                        farenheit_reported = farenheit_reported,
                        celcius_reported = celcius_reported))
```

Back to Example in Stan

- Results of inference

```
print(fit)
```

```
## Inference for Stan model: 75a5abc618a9cb52757beb282ebfbd0c.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##            mean se_mean   sd   2.5%   25%   50%   75% 97.5% n_eff Rhat
## true_temp 19.95    0.02 0.61  18.78 19.53 19.95 20.36 21.14  1412    1
## lp__      -9.77    0.02 0.74 -11.83 -9.95 -9.48 -9.28 -9.23  1671    1
##
## Samples were drawn using NUTS(diag_e) at Tue Feb 27 11:14:50 2018.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Main Approaches to Inference

- Conjugate priors
  - Set up the model so that the posterior distribution has a nice form

Main Approaches to Inference

- Conjugate priors
  - Set up the model so that the posterior distribution has a nice form
- Approximate Bayesian Computation (ABC)
  - Brute force inference by sampling according to $P(\mathbf{x}, \mathbf{z})$

Main Approaches to Inference

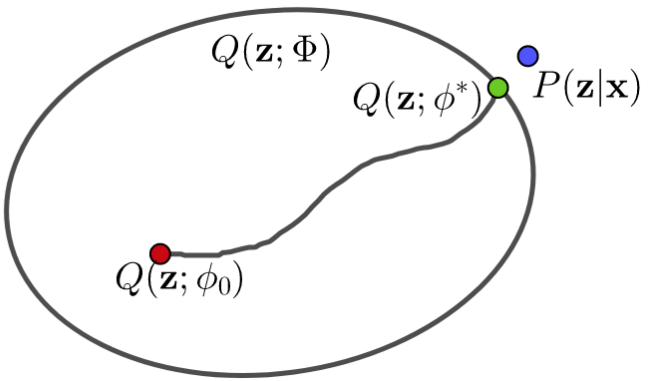
- Conjugate priors
  - Set up the model so that the posterior distribution has a nice form
- Approximate Bayesian Computation (ABC)
  - Brute force inference by sampling according to $P(\mathbf{x}, \mathbf{z})$
- Importance Sampling (More in Part II)
  - Choose a *tractable* distribution $Q(\mathbf{z})$ that approximates $P(\mathbf{z}|\mathbf{x})$
  - Compute a weighted average $f(\mathbf{z})$ over samples from Q
  - Weight for sample $\mathbf{z}_i$ = $w_i = \frac{P(\mathbf{x}=\mathbf{x}_o, \mathbf{z}_i)}{Q(\mathbf{z}_i)}$

Main Approaches to Inference

- Conjugate priors
  - Set up the model so that the posterior distribution has a nice form
- Approximate Bayesian Computation (ABC)
  - Brute force inference by sampling according to $P(\mathbf{x}, \mathbf{z})$
- Importance Sampling (More in Part II)
  - Choose a *tractable* distribution $Q(\mathbf{z})$ that approximates $P(\mathbf{z}|\mathbf{x})$
  - Compute a weighted average $f(\mathbf{z})$ over samples from Q
  - Weight for sample $\mathbf{z}_i$ = $w_i = \frac{P(\mathbf{x}=\mathbf{x}_o, \mathbf{z}_i)}{Q(\mathbf{z}_i)}$
- Markov Chain Monte Carlo (More in Part II)
  - Setup a Markov chain whose stationary distribution is $P(\mathbf{z}|\mathbf{x})$
  - Generate samples from the Markov chain and perform Monte Carlo integration
  - More efficient because correlated samples spend more time where the posterior likelihood is high

Main Approaches

- **Variational Inference** (Focus of this talk)
  - Choose a family of *tractable* distributions $Q(\mathbf{z}; \Phi)$ which contains a good approximation for $P(\mathbf{z}|\mathbf{x})$
  - Find a *good* match in the family $Q(\mathbf{z}; \phi^*)$
  - Integrate with respect to $Q(\mathbf{z}; \phi^*)$
  - If the family $Q(. , \Phi)$ is chosen appropriately, the integration can be done in closed form



  - Can be scaled to large datasets

Approximate Bayesian Computation (ABC)

- Basic Idea: Sample and filter
  - Sample from the model $\{\mathbf{x}_i, \mathbf{z}_i\}_{1,\ldots,K}$
  - Compute *sufficient statistics* $s(\mathbf{x}_i)$
  - Select samples $I \subset \{1, \ldots, K\}$ such that $s(\mathbf{x}_i) = s(\mathbf{x}_0)$ for $i \in I$
  - Compute

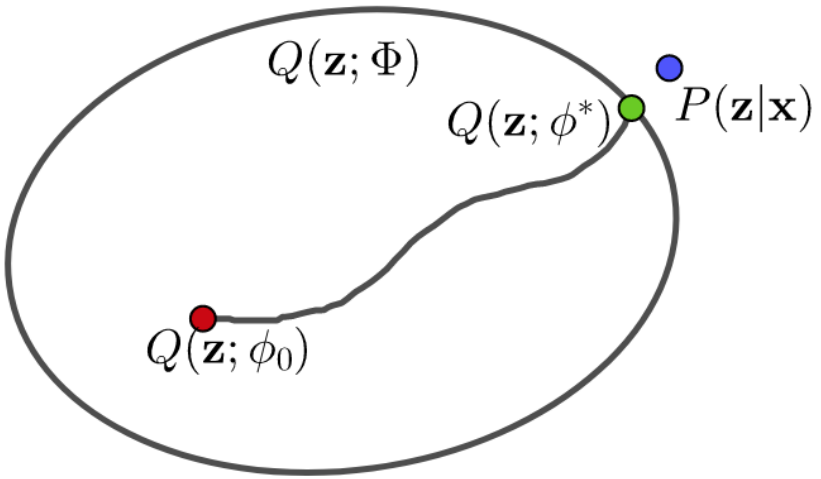
$$\mathrm{E}[f(\mathbf{z})|\mathbf{x} = \mathbf{x}_o] = \frac{1}{|I|} \sum_{i \in I} f(\mathbf{z}_i)$$

- Approximations
  - Sufficient statistics may be difficult to compute => approximate suff. statistics $\tilde{s}(\mathbf{x})$
  - Exact equality is likely to never occur => instead filter by $d(\tilde{s}(\mathbf{x}_i), \tilde{s}(\mathbf{x}_o)) < \epsilon$, where $d()$ is some distance function

Approximate Bayesian Computation (ABC)

- The bad
  - Computationally very inefficient for large models (many dimensions) or data
  - Too many rejections when the posterior is far from the prior
  - increasing ϵ increases bias
- The good
  - Is *likelihood-free* : Don't need a way to compute $P(\mathbf{x}, \mathbf{z})$
  - Embarassingly parallelizable

Variational Inference



- We will use the Kullback-Liebler (KL) divergence to measure the discrepancy between $Q(\mathbf{z};\phi)$ and $P(\mathbf{z}|\mathbf{x})$, which is given by

$$\mathrm{KL}(Q||P) = \int Q(\mathbf{z};\phi)\,\log\frac{Q(\mathbf{z};\phi)}{P(\mathbf{z}|\mathbf{x})}d\mathbf{z}$$

Variational Inference: The ELBO

- The quantity we want to minimize is

$$
\begin{aligned}
\mathrm{KL}(Q||P) &= \int Q(\mathbf{z};\phi)\,\log\frac{Q(\mathbf{z};\phi)}{P(\mathbf{z}|\mathbf{x})}d\mathbf{z} \\
&= \mathrm{E}_Q\left[\log\frac{Q(\mathbf{z};\phi)}{P(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathrm{E}_Q\left[\log(Q(\mathbf{z};\phi)) - \log(P(\mathbf{z}|\mathbf{x}))\right] \\
&= \mathrm{E}_Q\left[\log(Q(\mathbf{z};\phi)) - \log(P(\mathbf{x},\mathbf{z}))\right] + \mathrm{E}_Q[\log(P(\mathbf{x}))] \\
&= \underbrace{-\mathrm{E}_Q[\log(P(\mathbf{x},\mathbf{z})) - \log(Q(\mathbf{z};\phi))]}_{\text{ELBO}} + \underbrace{\log(P(\mathbf{x}))}_{\text{Evidence}}
\end{aligned}
$$

Variational Inference: The ELBO

- The quantity we want to minimize is

$$\begin{aligned}
\mathrm{KL}(Q||P) &= \int Q(\mathbf{z};\phi)\,\log\frac{Q(\mathbf{z};\phi)}{P(\mathbf{z}|\mathbf{x})}d\mathbf{z} \\
&= \mathrm{E}_Q\left[\log\frac{Q(\mathbf{z};\phi)}{P(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathrm{E}_Q\left[\log(Q(\mathbf{z};\phi)) - \log(P(\mathbf{z}|\mathbf{x}))\right] \\
&= \mathrm{E}_Q\left[\log(Q(\mathbf{z};\phi)) - \log(P(\mathbf{x},\mathbf{z}))\right] + \mathrm{E}_Q[\log(P(\mathbf{x}))] \\
&= \underbrace{-\mathrm{E}_Q[\log(P(\mathbf{x},\mathbf{z})) - \log(Q(\mathbf{z};\phi))]}_{\text{ELBO}} + \underbrace{\log(P(\mathbf{x}))}_{\text{Evidence}}
\end{aligned}$$

- Because $\mathrm{KL}(Q||P) \geq 0$,

$$\mathrm{E}_Q[\log(P(\mathbf{x},\mathbf{z})) - \log(Q(\mathbf{z};\phi))] \leq \log(P(\mathbf{x}))$$

- Evidence Lower Bound (ELBO) is the quantity that we *maximize* to minimize discrepancy between the variational distribution and the posterior

The ELBO

- Why is the ELBO easier to work with?
  - We don't need to compute the pesky normalization term $P(\mathbf{x})$
  - The ELBO can be estimated by averaging over samples $\mathbf{z}_i$ from Q

$$\text{ELBO} = \frac{1}{K}\sum_{i}^{K} \log(P(\mathbf{x} = \mathbf{x}_o, \mathbf{z}_i)) - \log(Q(\mathbf{z}_i; \phi))$$

The ELBO

- Why is the ELBO easier to work with?
  - We don't need to compute the pesky normalization term $P(\mathbf{x})$
  - The ELBO can be estimated by averaging over samples $\mathbf{z}_i$ from Q

$$\text{ELBO} = \frac{1}{K}\sum_{i}^{K} \log(P(\mathbf{x} = \mathbf{x}_o, \mathbf{z}_i)) - \log(Q(\mathbf{z}_i; \phi))$$

- How do we maximize the ELBO w.r.t. ϕ?
  - Stochastic Gradient Descent
  - Stochastic Variational Inference (SVI)
- We need an unbiased estimator of the gradient

Gradient estimator 1: Score function gradient

- Use the score function (log-derivative) trick
- Estimate the score function gradient from samples from Q

$$\begin{aligned}\nabla_\phi \text{ELBO} &= \nabla_\phi \text{E}_Q[\log(P(\mathbf{x}, \mathbf{z})) - \log(Q(\mathbf{z}; \phi))] \\ &= \text{E}_Q[(\log(P(\mathbf{x}, \mathbf{z})) - \log(Q(\mathbf{z}; \phi)))\nabla_\phi \log(Q(\mathbf{z}; \phi))] \\ &\approx \frac{1}{K}\sum_i^K (\log(P(\mathbf{x}, \mathbf{z}_i)) - \log(Q(\mathbf{z}_i; \phi)))\nabla_\phi \log(Q(\mathbf{z}_i; \phi))\end{aligned}$$

Gradient Estimator 2: The Reparameterization Trick

- Some families $Q(\mathbf{z}, \Phi)$ can be written as

$$\epsilon \sim Q(\epsilon)$$
$$\mathbf{z} = g(\epsilon, \phi)$$

Gradient Estimator 2: The Reparameterization Trick

- Some families $Q(\mathbf{z}, \Phi)$ can be written as

$$\epsilon \sim Q(\epsilon)$$
$$\mathbf{z} = g(\epsilon, \phi)$$

  - Examples

$Q(\mathbf{z}, \phi)$	$Q(\epsilon)$	$g(\epsilon, \phi)$
$\mathbf{z} \sim \text{Normal}(\mu, \sigma)$	$\epsilon \sim \text{Normal}(0, 1)$	$\mathbf{z} = \sigma\epsilon + \mu$
$\mathbf{z} \sim \text{Log-normal}(\mu, \sigma)$	$\epsilon \sim \text{Normal}(0, 1)$	$\mathbf{z} = \exp(\sigma\epsilon + \mu)$
$\mathbf{z} \sim \text{f}_\phi$	$\epsilon \sim \text{Uniform}(0, 1)$	$\mathbf{z} = \text{F}_\phi^{-1}(\epsilon)$

Gradient Estimator 2: The Reparameterization Trick

- For reparameterizable Q we can write the gradient of the ELBO as

$$\begin{aligned}\nabla_\phi \mathrm{ELBO} &= \nabla_\phi \mathrm{E}_{Q(\mathbf{z})}[\log(P(\mathbf{x},\mathbf{z})) - \log(Q(\mathbf{z};\phi))] \\ &= \nabla_\phi \mathrm{E}_{Q(\epsilon)}[\log(P(\mathbf{x},g(\epsilon,\phi))) - \log(Q(g(\epsilon,\phi)))] \\ &= \mathrm{E}_{Q(\epsilon)}[\nabla_\phi \log(P(\mathbf{x},g(\epsilon,\phi))) - \nabla_\phi \log(Q(g(\epsilon,\phi)))]\end{aligned}$$

- This estimate of the gradient generally has lower variance than the score function gradient estimate

Back to the Thermometer Problem

- Let us make the problem a bit harder
- Assume that the affine transformation between Celcius and Farenheit is unknown
- So, given noisy samples from the Celcius and Farenheit thermometers we want to estimate
    - The poseterior mean and variance of the true temeperature in Celcius
    - The multiplier and shift parameters of the Farenheit scale

Programming the Model in PyRo

```
# The forward model
def temperature_model(celcius_reported, farenheit_reported, celcius_sigma, farenhe
    multiplier = pyro.param("multiplier", Variable(torch.Tensor([1]), requires_gra
    shift = pyro.param("shift", Variable(torch.Tensor([25]), requires_grad=True))
    # true_temp is a prior assumed to be in [15, 50]
    true_temp = pyro.sample("true_temp", dist.uniform,
                            Variable(torch.Tensor([15])),
                            Variable(torch.Tensor([50])))
    for i in pyro.irange("locals_celcius", len(celcius_reported), subsample_size=3
        pyro.sample("celcius_obs_{}".format(i), dist.normal, true_temp,
                    Variable(torch.Tensor([celcius_sigma])),
                    obs=Variable(torch.Tensor([celcius_reported[i]])))
    for i in pyro.irange("locals_farenheit", len(farenheit_reported), subsample_si
        pyro.sample("farenheit_obs_{}".format(i), dist.normal, true_temp * multipl
                    Variable(torch.Tensor([farenheit_sigma])),
                    obs=Variable(torch.Tensor([farenheit_reported[i]])))
```

```
# Define the Variational Family
def Q(celcius_reported, farenheit_reported, celcius_sigma, farenheit_sigma):
    true_temp_mean = pyro.param("true_temp_mean", Variable(torch.Tensor([25]), req
    true_temp_log_sigma = pyro.param("true_temp_log_sigma", Variable(torch.Tensor(
    pyro.sample("true_temp", dist.normal, true_temp_mean, torch.exp(true_temp_log_
```

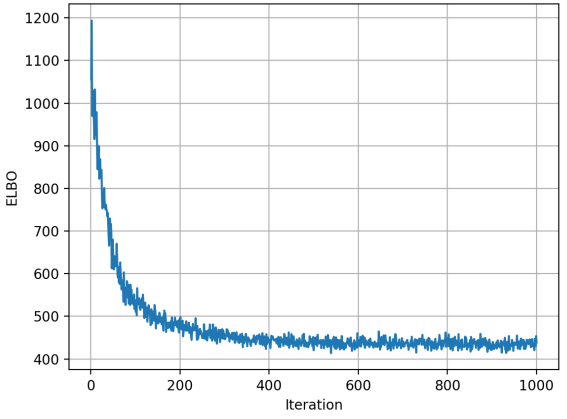
Variational Inference in PyRo

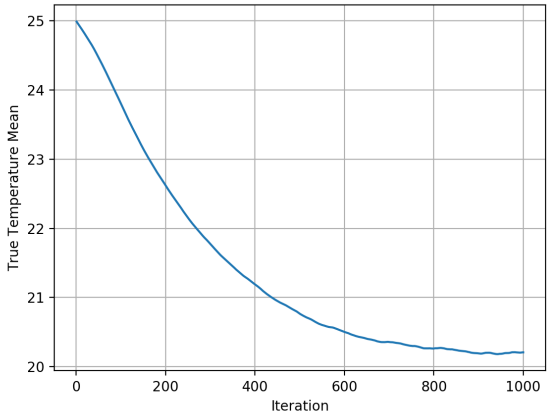
- Generate some data and perform inference

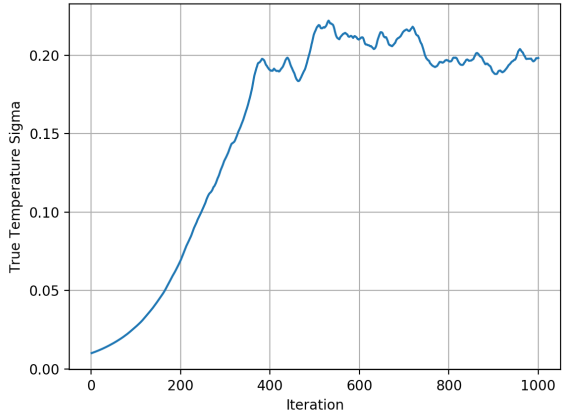
```
def main():
    true_temp = 20 #unknown
    celcius_sigma = 2.0  # known
    farenheit_sigma = 6.0  # known
    Nc =  50  # size of the data
    Nf =  100  # size of the data
    celcius_reported = np.random.normal(0, 1, Nc) * celcius_sigma + true_temp
    farenheit_reported  = np.random.normal(0, 1, Nf) * farenheit_sigma + true_temp

    adam_params = {"lr": 0.05, "betas": (0.95, 0.99)}
    optimizer = Adam(adam_params)
    svi = SVI(temperature_model, Q, optimizer, loss="ELBO", num_particles=20)
    losses = []
    for step in range(1000):
        losses.append(svi.step(celcius_reported, farenheit_reported, celcius_sigma
        a = pyro.get_param_store()
        print(a._params)
```

PyRo Thermometer Inference Results

Variable	Value
True temperature mean	20.2
True temperature Sigma	0.2
Unknown Multiplier	2.1
Unknown Shift	26.2



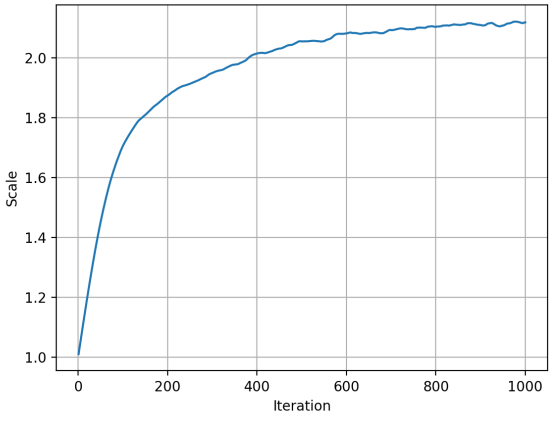


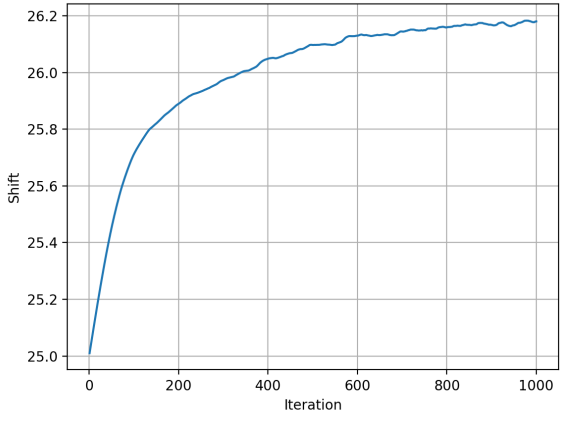


PyRo Thermometer Inference Results

Variable	Value
True temperature mean	20.2
True temperature Sigma	0.2
Unknown Multiplier	2.1
Unknown Shift	26.2

- The temperature scale is *not identifiable*
  - $20 \times 1.8 + 32 \approx 20 \times 2.1 + 26.2$





Scaling to Big Data

- Many big data models have the following form of conditional independence

$$
\begin{aligned}
&\text{for } \; i \in 1, \ldots, N \\
&\quad \mathbf{z}_i \sim P(\mathbf{z}|\theta) \\
&\quad \mathbf{x}_i \sim P(\mathbf{x}|f(\mathbf{z}_i, \lambda))
\end{aligned}
$$

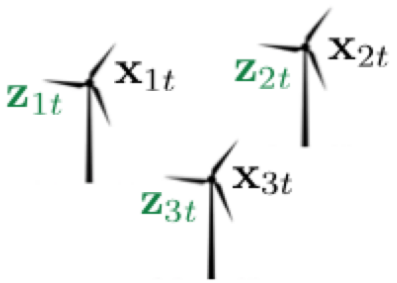
Scaling to Big Data

- Many big data models have the following form of conditional independence

$$\begin{aligned} &\text{for } \; i \in 1, \ldots, N \\ &\quad \mathbf{z}_i \sim P(\mathbf{z}|\theta) \\ &\quad \mathbf{x}_i \sim P(\mathbf{x}|f(\mathbf{z}_i, \lambda)) \end{aligned}$$

- Example

$$\begin{aligned} \text{Unobserved free-stream wind-speeds at time } t : \mathbf{z}_t &\triangleq [\mathbf{z}_{1t}, \mathbf{z}_{2t}, \mathbf{z}_{3t}] \\ \text{Observed waked wind-speeds at time } t : \mathbf{x}_t &\triangleq [\mathbf{x}_{1t}, \mathbf{x}_{2t}, \mathbf{x}_{3t}] \\ \text{Paramerized wake model : } f(.,\lambda) & \end{aligned}$$



$$\begin{aligned} \mathbf{z}_{it} &\sim \text{Weibull}(\alpha_i, \beta_i) \\ \mathbf{x}_t &\sim \text{Normal}(f(\mathbf{z}_t, \lambda), \epsilon I) \end{aligned}$$

Scaling Computationally: Sampling

- When the models have the above type of conditional independence $\log P(\mathbf{x}_i, \mathbf{z}_i)$ can be written as

$$\begin{aligned}\log P(\mathbf{x}, \mathbf{z}) &= \sum_{i}^{N} \mathbf{log} P(\mathbf{x}_i, \mathbf{z}_i) \\ &\approx \frac{N}{M} \sum_{i \in I_M} \mathbf{log} P(\mathbf{x}_i, \mathbf{z}_i)\end{aligned}$$

Scaling Computationally: Sampling

- When the models have the above type of conditional independence $\log P(\mathbf{x}_i, \mathbf{z}_i)$ can be written as

$$\begin{aligned}\log P(\mathbf{x}, \mathbf{z}) &= \sum_i^N log P(\mathbf{x}_i, \mathbf{z}_i) \\ &\approx \frac{N}{M} \sum_{i \in I_M} log P(\mathbf{x}_i, \mathbf{z}_i)\end{aligned}$$

- If our variational family is also chosen so that $Q(\mathbf{z}, \phi) = \prod_i^N Q(\mathbf{z}_i, \phi)$
  - Then both ELBO and $\nabla_\phi \mathrm{ELBO}$ can be estimated on a subset of the data (by appropriate scaling)
  - The estimates are still unbiased but with increased variance
- Similar to mini-batch Stochastic Gradient Descent

Scaling Statistically: Amortized Inference

- In the wake estimation problem we are interested in the posterior distribution of the unwaked wind-speeds $\mathbf{z}$ given the observed waked wind-speeds $\mathbf{x}$, i.e. $P(\mathbf{z}_{1,\ldots N}|\mathbf{x}_{1,\ldots,N})$.

- Since we expect that the posterior distribution will be different for different time-stamps we might choose the variational family of the form

$$Q(\mathbf{z}_i, \phi_i)$$

Scaling Statistically: Amortized Inference

- In the wake estimation problem we are interested in the posterior distribution of the unwaked wind-speeds $\mathbf{z}$ given the observed waked wind-speeds $\mathbf{x}$, i.e. $P(\mathbf{z}_{1,\ldots N}|\mathbf{x}_{1,\ldots,N})$.
- Since we expect that the posterior distribution will be different for different time-stamps we might choose the variational family of the form

$$Q(\mathbf{z}_i, \phi_i)$$

- The problem with this choice is that the number of parameters scales with size of data

Scaling Statistically: Amortized Inference

- In the wake estimation problem we are interested in the posterior distribution of the unwaked wind-speeds $\mathbf{z}$ given the observed waked wind-speeds $\mathbf{x}$, i.e. $P(\mathbf{z}_{1,\ldots N}|\mathbf{x}_{1,\ldots,N})$.
- Since we expect that the posterior distribution will be different for different time-stamps we might choose the variational family of the form

$$Q(\mathbf{z}_i, \phi_i)$$

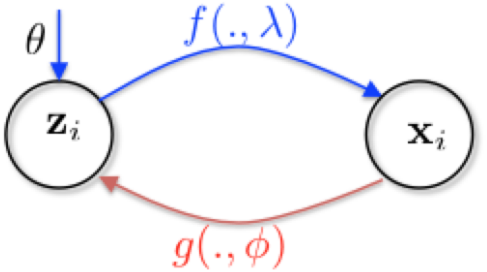
- The problem with this choice is that the number of parameters scales with size of data
- Since we know that z_i is predictable from x_i, we could pick a variational family of the form

$$Q(\mathbf{z}_i, g(\mathbf{x}_i, \phi))$$

  - $g(.)$ might be a deep neural-network
- The burden of estimating ϕ is *amortized* over the entire data set

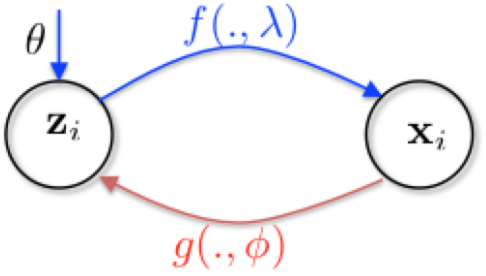
Variational Auto Encoder

- The model along with the amortized variational family can be depicted as

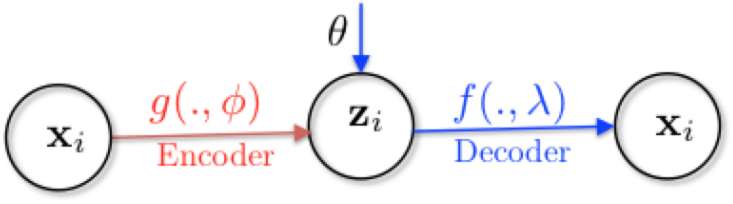


Variational Auto Encoder

- The model along with the amortized variational family can be depicted as



- This can be redrawn as



- When both f and g are parameterized deep neural networks, and the dimension of $\mathbf{z}$ is smaller than that of $\mathbf{x}$, this architechture can be used for compression and to learn to sample from complex distributions
- This model is called the Variational Auto Encoder (VAE)

Other Topics for the Interested

- Expectation propagation
- Mean-field variational inference
- Variance reduction for ELBO gradient estimation
    - Control variates
    - Rao-Blackwellization
- Black-box variational inference
- Likelihood-free variational inference
- Generalized reparameterized gradient
- Stochastic computational graphs

References

- *Pattern Recognition and Machine Learning*, C. Bishop, Chapter 10
- *Variational Inference: A Review for Statisticians*, D. M. Blei, A. Kucukelbir, J. D. McAuliffe
- *Automated Variational Inference in Probabilistic Programming*, D. Wingate, T. Weber
- *Auto-Encoding Variational Bayes*, D. P. Kingma, M. Welling