AI Programming: Lecture 4

Variational Inference

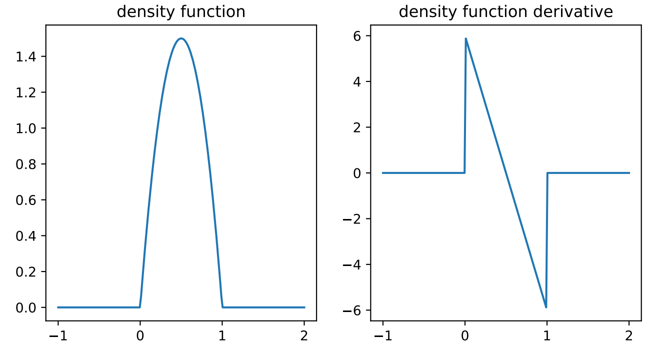
Markus Böck and Jürgen Cito

Research Unit of Software Engineering

Table of contents

From Constrained to Unconstrained Model

From Constrained to Unconstrained Model



From Constrained to Unconstrained Model

Every density that is defined for constrained continuous variables can be transformed to an unconstrained density. For $Y = H(X)$

$$p_Y(y) = p_X(H^{-1}(y)) \cdot \left|\frac{d}{dy}H^{-1}(y)\right| \qquad \text{defined for } y \in H^{-1}(\text{support}(X))$$

$$p_X(x) = p_Y(H(x)) \cdot \left|\frac{d}{dx}H(x)\right| \qquad \text{defined for } x \in \text{support}(X)$$

From Constrained to Unconstrained Model

Example:

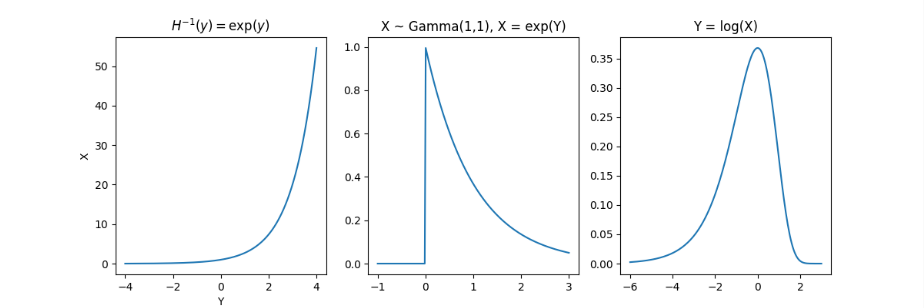
$X \sim \text{Gamma}(1, 1)$ is supported on $[0, \infty)$

Choose $y = H(x) = \log(x)$

$$p_Y(y) = p_X(\exp(y)) \cdot \exp(y)$$

is supported on $\mathbb{R}$.

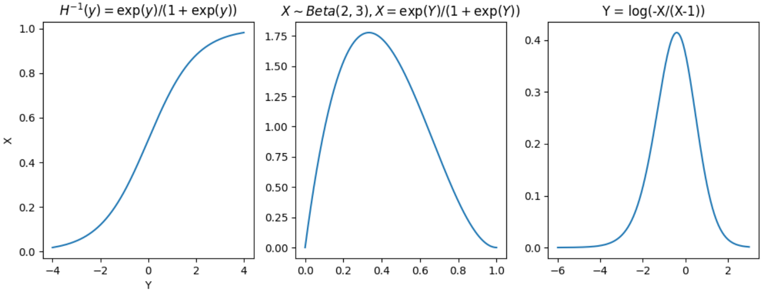
From Constrained to Unconstrained Model



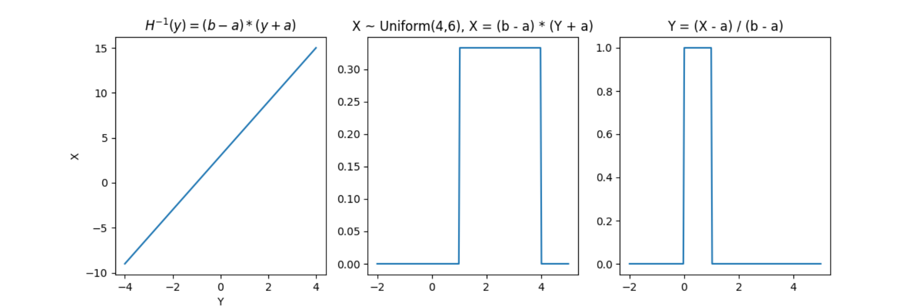
With pytorch:

```
T = dist.transform_to(dist_constrained.support)
dist_unconstrained = dist.TransformedDistribution(dist_constrained, T.inv)
```

From Constrained to Unconstrained Model



From Constrained to Unconstrained Model

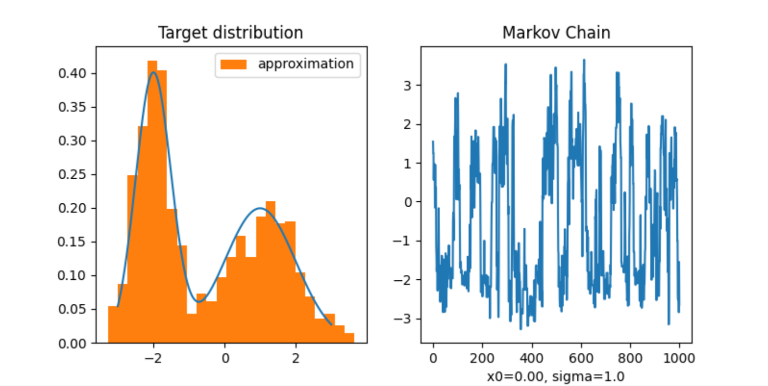


Variational Inference

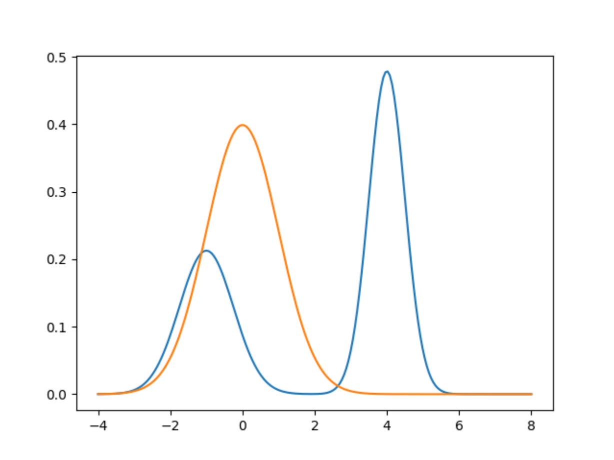
Variational Inference

- Assumption: Finite number of continuous (unconstrained) random variables
- Idea: Approximate the posterior with a fixed-form parameterised variational distribution.
- Optimise the parameters of the variational distribution to be ”close” to true posterior.

Variational Inference



Variational Inference



Adjust mean and deviation of orange Normal distribution to ”fit” blue distribution.

How to measure distance between two distributions?

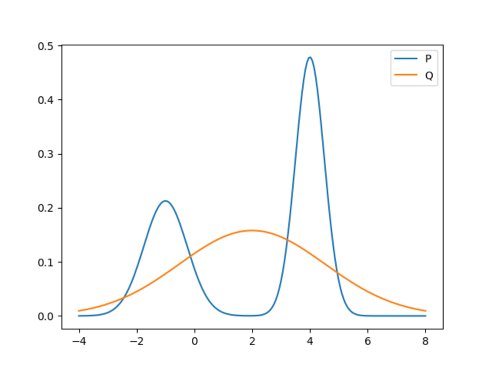
Kullback-Leibler Divergence

$$\begin{aligned} D_{\mathrm{KL}}(P \parallel Q) &= \mathbb{E}_{X\sim P}\left[\log\left(\frac{P(X)}{Q(X)}\right)\right] \\ &= \underbrace{\mathbb{E}_{X\sim P}\left[\log P(X)\right]}_{=-\text{entropy}} - \mathbb{E}_{X\sim P}\left[\log Q(X)\right] \end{aligned}$$

It is the amount of information lost when Q is used to approximate P.

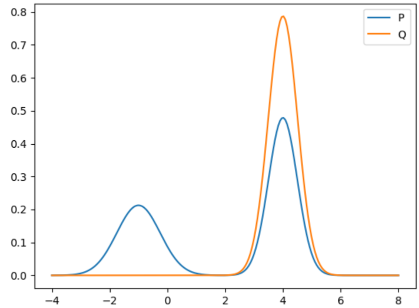
Variational Inference: Kullback Leibler Divergence

$$\min_Q D_{\mathrm{KL}}(P \parallel Q) = \min_Q \mathbb{E}_{X \sim P}\left[\log\left(\frac{P(X)}{Q(X)}\right)\right]$$



Variational Inference: Kullback Leibler Divergence

$$\min_{Q} D_{\mathrm{KL}}(Q \parallel P) = \min_{Q} \mathbb{E}_{X \sim Q}\left[\log\left(\frac{Q(X)}{P(X)}\right)\right]$$



KL-Divergence is not symmetric!

Variational Inference: Kullback Leibler Divergence

How to compute $D_{\mathrm{KL}}(P \parallel Q)$?

- Discrete Variables:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x} P(x) \log\left(\frac{Q(x)}{P(x)}\right)$$

- Continuous Variables:

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} P(x) \log\left(\frac{Q(x)}{P(x)}\right) dx$$

- Sampling!

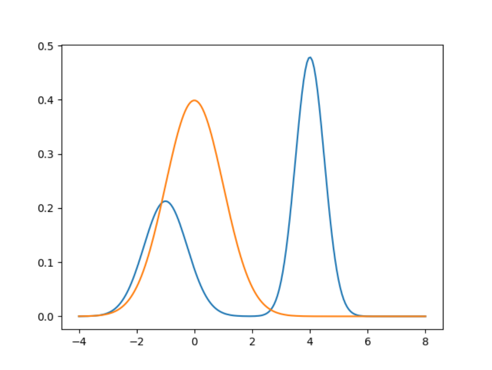
$$D_{\mathrm{KL}}(P \parallel Q) \approx \frac{1}{N} \sum_{i=1}^{N} \log\left(\frac{Q(x_i)}{P(x_i)}\right) \quad x_i \sim P$$

Variational Inference: Kullback Leibler Divergence

```python
# numerical integration
def integrate(func: Callable[[torch.Tensor], torch.Tensor], a, b, N=500):
    x = torch.linspace(a, b, N)
    y = func(x)
    return torch.trapz(y, x)
def kl_divergence_1(p: dist.Distribution, q: dist.Distribution, a, b, N=500):
    def integrand(x: torch.Tensor) -> torch.Tensor:
        px = p.log_prob(x).exp()
        qx = q.log_prob(x).exp()
        I = px * torch.log(px / qx)
        I[px == 0] = 0
        return I
    return integrate(integrand, a, b, N)


# sampling
def kl_divergence_2(p: dist.Distribution, q: dist.Distribution, N=500):
    x = p.sample((N,))
    px = p.log_prob(x).exp()
    qx = q.log_prob(x).exp()
    I = torch.log(px / qx)
    return I.mean()
```

Variational Inference: Kullback Leibler Divergence




```
kl_divergence_1(normal_mixture, normal, -5, 7, N=10000)
```
Returns:
```
tensor(4.5455)
```

```
kl_divergence_2(normal_mixture, normal, N=1000000)
```
Returns:
```
tensor(4.5453)
```

Variational Inference: Kullback Leibler Divergence

For Bayesian inference $P = P(\Theta|X)$.

We cannot sample from posterior, so we are bound to $D_{\mathrm{KL}}(Q \parallel P) = \mathbb{E}_{\Theta\sim Q}\left[\log\left(\frac{Q(\Theta)}{P(\Theta|X)}\right)\right]$, where we sample from the variational distribution.

We cannot evaluate $P(\Theta|X)$ and we rewrite to joint.

$$
\begin{aligned}
D_{\mathrm{KL}}(Q \parallel P) &= \mathbb{E}_{\Theta\sim Q}\left[\log\left(\frac{Q(\Theta)}{P(\Theta|X)}\right)\right] = \mathbb{E}_{\Theta\sim Q}\left[\log\left(\frac{Q(\Theta)P(X)}{P(\Theta,X)}\right)\right] \\
&= \mathbb{E}_{\Theta\sim Q}\left[\log Q(\Theta) - \log P(\Theta,X)\right] + \mathbb{E}_{\Theta\sim Q}\left[\log P(X)\right] \\
&= -\underbrace{\mathbb{E}_{\Theta\sim Q}\left[\log P(\Theta,X) - \log Q(\Theta)\right]}_{\mathrm{ELBO}(Q,P)} + \underbrace{\log P(X)}_{\text{log-evidence}} \\
\underbrace{\log P(X)}_{\text{constant}} &= D_{\mathrm{KL}}(Q \parallel P) + \mathrm{ELBO}(Q,P)
\end{aligned}
$$

Maximising the ELBO w.r.t. Q minimizes $D_{\mathrm{KL}}(Q \parallel P)$!

Variational Inference: Example

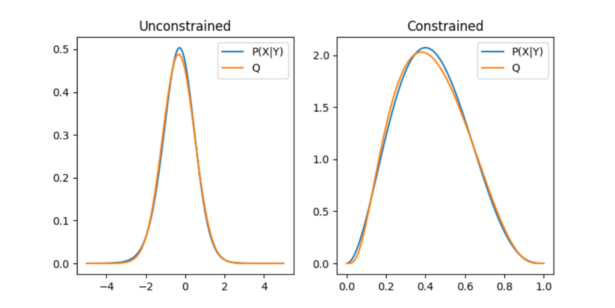
Coin model:

```
flips = torch.tensor([0.,1.,1.,0.,0.])
prior = dist.Uniform(0,1)
posterior = dist.Beta(1+flips.sum(), 1+(1-flips).sum()) # analytical solution
def likelihood(p):
    return dist.Bernoulli(p).log_prob(flips.reshape(-1,1)).sum(dim=0).exp()

# transform to unconstrained model
T = dist.transforms.transform_to(posterior.support) # sigmoid: R -> [0,1]
posterior_unconstrained = dist.TransformedDistribution(posterior, T.inv)
prior_uncontrained = dist.TransformedDistribution(prior, T.inv)
```

Variational Inference: Example

```
def DKL_QP(X):
    # variational distribution is Normal parameterised with mean and sigma
    mu, sigma = X
    q = dist.Normal(mu, sigma)
    return kl_divergence(q, posterior_unconstrained, mu-4*sigma, mu+4*sigma)
# minimize by brute force (grid search)
mu, sigma = brute(DKL_QP, [(-3,3), (0.1,1.)], Ns=100)
variational = dist.Normal(mu, sigma)
```



Variational Inference

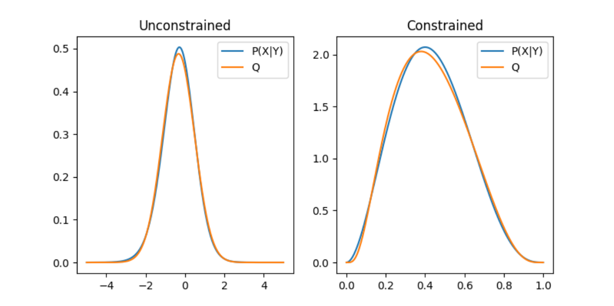
Why don't we fit Gaussian to constrained posterior?

$$D_{\mathrm{KL}}(Q \parallel P) = \mathbb{E}_{\Theta \sim Q}\left[\log\left(\frac{Q(\Theta)}{P(\Theta|X)}\right)\right]$$

Only well-defined if $P(\theta|X) = 0 \implies Q(\theta) = 0$.

Variational Inference: Example

```
def ELBO_loss(X):
    mu, sigma = X
    q = dist.Normal(mu, sigma)
    def integrand(x: torch.Tensor) -> torch.Tensor:
        px = likelihood(T(x)) * prior_uncontrained.log_prob(x).exp()
        qx = q.log_prob(x).exp()
        I = qx * torch.log(px / qx)
        I[px == 0] = 0
        return I
    return -integrate(integrand, mu-4*sigma, mu+4*sigma) # -ELBO
mu, sigma = brute(ELBO_loss, [(-3,3), (0.1,1.)], Ns=100) # minimize
```

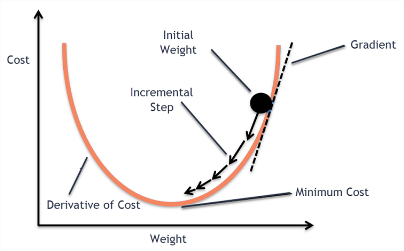


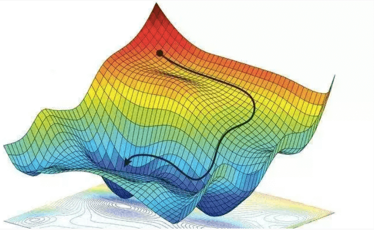
Automatic Differentiation Variational Inference (ADVI)

ADVI

- Computing D_{KL} with numerical integration $\rightarrow$ estimate with sampling
- Optimise parameters with grid-search $\rightarrow$ perform gradient descent
- Single variable $\rightarrow$ multiple variables

ADVI: Gradient Descent





ADVI: Automatic Differentiation (AD)

```
X = torch.tensor(2., requires_grad=True)

Y = X**2 + torch.log(X)

Y.backward() # compute gradients

X.grad.item(), (2*X + 1/X).item()
```

Returns:
```
(4.5, 4.5)
```

ADVI: Variational Distribution

Mean-field Gaussian:

Independent Gaussian for each of K variables:

$$Q(\boldsymbol{\theta}; \phi) = \mathrm{MvNormal}(\boldsymbol{\theta}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)) = \prod_{k=1}^{K} \mathrm{Normal}(\theta_k; \mu_k, \sigma_k^2)$$

Q parameterised by unconstrained $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{\omega})$ with $\boldsymbol{\sigma} = \exp(\boldsymbol{\omega})$.

Full-rank Gaussian:

Correlated Gaussians

$$Q(\boldsymbol{\theta}; \phi) = \mathrm{MvNormal}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Q parameterised by unconstrained $\boldsymbol{\phi} = (\boldsymbol{\mu}, L)$ with $\boldsymbol{\Sigma} = LL^T$.

ADVI: ELBO

Assumption:

- Unconstrained model for simpler notation.
- Mean-field Gaussian: $Q(\boldsymbol{\theta}; \boldsymbol{\phi}) = \text{MvNormal}(\boldsymbol{\theta}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$

$$\text{ELBO}(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{\theta} \sim Q(.;\boldsymbol{\phi})} [\log P(\boldsymbol{\theta}, X)] + \underbrace{\mathbb{E}_{\boldsymbol{\theta} \sim Q(.;\boldsymbol{\phi})} [-\log Q(\boldsymbol{\theta}; \boldsymbol{\phi})]}_{\text{Entropy of } Q}$$

Entropy can be determined analytically

$$\mathbb{H}[Q(\boldsymbol{\theta}; \boldsymbol{\phi})] = \frac{K}{2} \log(2\pi e) + \frac{1}{2} \sum_{k=1}^{K} \log \sigma_k^2$$

ADVI: ELBO Gradient: Reparametrisation Trick

$$
\begin{aligned}
\nabla_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta} \sim Q(.;\boldsymbol{\phi})} [\log P(\boldsymbol{\theta}, X)] &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta} \sim \text{MvNormal}(\boldsymbol{\mu}, \text{diag}(\exp(\boldsymbol{\omega})^2))} [\log P(\boldsymbol{\theta}, X)] \\
&= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\eta} \sim \text{MvNormal}(\mathbf{0}, I)} [\log P(\exp(\boldsymbol{\omega}) \cdot \boldsymbol{\eta} + \boldsymbol{\mu}, X)] \\
&= \mathbb{E}_{\boldsymbol{\eta} \sim \text{MvNormal}(\mathbf{0}, I)} \Big[\underbrace{\nabla_{\boldsymbol{\phi}} \log P(\exp(\boldsymbol{\omega}) \cdot \boldsymbol{\eta} + \boldsymbol{\mu}, X)}_{\text{Compute with AD}} \Big]
\end{aligned}
$$

Putting all together:

For $\boldsymbol{\eta}_i \sim \text{MvNormal}(\mathbf{0}, I)$

$$
\begin{aligned}
\nabla_{\boldsymbol{\mu}} \text{ELBO}(\phi) &\approx \frac{1}{L} \sum_{i=1}^{L} \nabla_{\boldsymbol{\mu}} \log P(\exp(\boldsymbol{\omega}) \cdot \boldsymbol{\eta}_i + \boldsymbol{\mu}, X) \\
\nabla_{\boldsymbol{\omega}} \text{ELBO}(\phi) &\approx \frac{1}{L} \sum_{i=1}^{L} \nabla_{\boldsymbol{\omega}} \log P(\exp(\boldsymbol{\omega}) \cdot \boldsymbol{\eta}_i + \boldsymbol{\mu}, X) \qquad \underbrace{+1}_{\text{Derivative of entropy}}
\end{aligned}
$$

Typically, $1 \leq L \leq 10$.

ADVI: constrained case

Transformation

$$T : \text{support}(P(\theta)) \rightarrow \mathbb{R}^K, \quad \boldsymbol{\zeta} = T(\boldsymbol{\theta})$$

The transformed density becomes

$$P(\zeta, X) = P(T^{-1}(\zeta))|\det J_{T^{-1}}(\zeta)|$$

and we fit the Gaussians to the unconstrained density

$$\text{ELBO}(\phi) = \mathbb{E}_{\boldsymbol{\zeta} \sim Q(.;\phi)} \left[\log P(T^{-1}(\zeta), X) + \log |\det J_{T^{-1}}(\zeta)| \right] + \underbrace{\mathbb{H}\left[Q(\zeta; \phi)\right]}_{\text{Entropy of } Q}$$

```
T = dist.transforms.transform_to(dist_constrained.support)
dist_unconstrained = dist.TransformedDistribution(dist_constrained, T.inv)
```

ADVI

1. Initialise $\boldsymbol{\phi}$ (e.g. $\boldsymbol{\phi} = \mathbf{0}$)
2. Repeat
   - Draw L samples $\boldsymbol{\eta}_i \sim \text{MvNormal}(\mathbf{0}, \boldsymbol{I})$
   - Approximate gradient $\nabla_{\boldsymbol{\phi}} \text{ELBO}(\boldsymbol{\phi})$ with $\boldsymbol{\eta}_i$.
   - Update $\boldsymbol{\phi}$ with gradient (e.g. Adagrad update rule)

Hard math - easier implementation.

See Assignment 4.

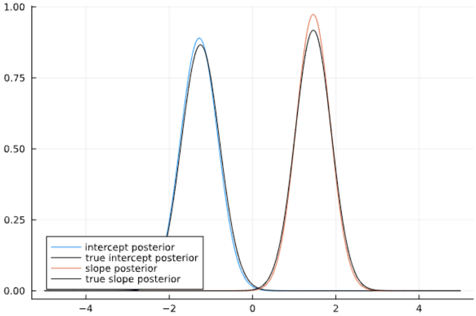
ADVI in Turing (Linear Regression)

```
model = linear_regression(x,y; broad_prior_params...);
```

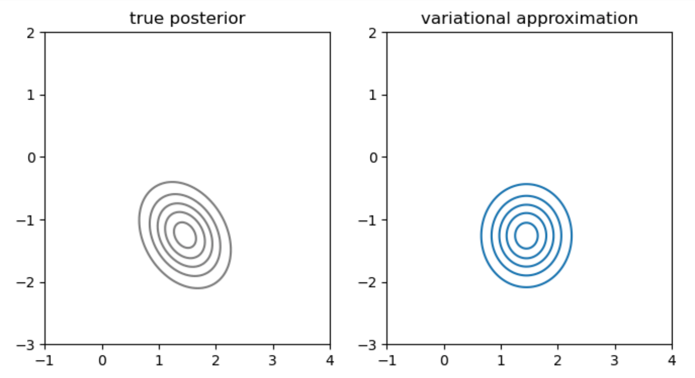
```
advi = ADVI(25, 10_000); # (samples_per_step, max_iters)
Q = Variational.meanfield(model); # assumption: posterior distribution consists of Normal distributions
result = vi(model, advi, Q; optimizer=Variational.DecayedADAGrad(1e-2, 1.1, 0.9))
```

```
[ADVI] Optimizing... 100% Time: 0:00:01
MultivariateTransformed{DistributionsAD.TuringDiagMvNormal{Vector{Float64}, Vector{Float64}}, Stacked{Vecto
dist: DistributionsAD.TuringDiagMvNormal{Vector{Float64}, Vector{Float64}}(m=[1.459082954777685, -1.2795708
transform: Stacked([identity, identity], UnitRange{Int64}[1:1, 2:2], UnitRange{Int64}[1:1, 2:2])
)
```

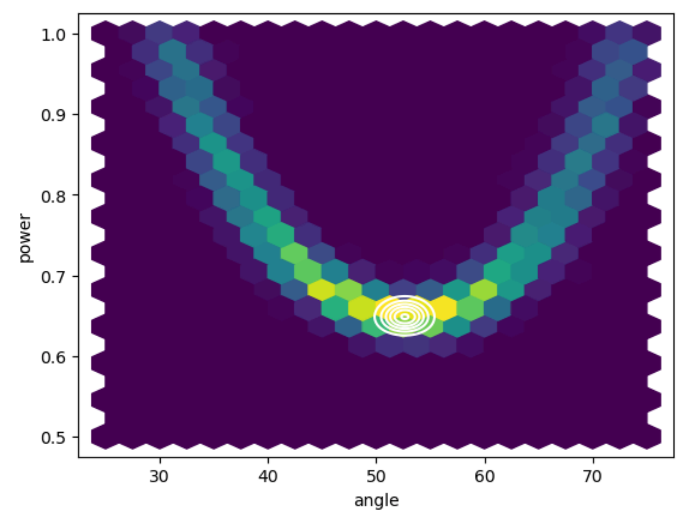
```
plot(x -> pdf(Normal(result.dist.m[2], result.dist.σ[2]), x), label="intercept posterior", color=1)
plot!(x -> pdf(true_intercept_posterior, x), label="true intercept posterior", color="black")
plot!(x -> pdf(Normal(result.dist.m[1], result.dist.σ[1]), x), label="slope posterior", color=2)
plot!(x -> pdf(true_slope_posterior, x), label="true slope posterior", color="black", legend=:bottomleft)
```



ADVI in Turing (Linear Regression)



ADVI in Turing (Golf)



ADVI in Pyro

https://pyro.ai/examples/bayesian_regression_ii.html

https://pyro.ai/examples/vae.html

Stochastic Variational Inference

Stochastic Variational Inference

The complexity of ADVI is $\mathcal{O}(NLK)$, where N is the number of data points, since

$$\log P(\Theta, X) = \sum_{i=1}^{N} \log P(\Theta, X_i).$$

Thus, ADVI as presented, does not scale well to large datasets.

Subsampling

Suppose we have X_j are independent and identically distributed (iid). We are interested in the quantity

$$\sum_{j=1}^{N} f(X_j).$$

Draw an index uniformly at random $J \sim \text{DiscreteUniform}(1, \dots, N)$, then

$$\sum_{j=1}^{N} f(X_j) = N \cdot \mathbb{E}_J \left[f(X_J) \right].$$

The rightt hand side can be approximated with the Monte Carlo method, $j_m \sim \text{DiscreteUniform}(1, \dots, N)$ iid,

$$\sum_{j=1}^{N} f(X_j) \approx \frac{N}{M} \sum_{m=1}^{M} f(X_{j_m}).$$

Subsampling

Thus, from a large dataset we can subsample a mini-batch by selecting data points at random, $\{X_{j_1}, \ldots, X_{j_M}\}$, and the sum will be approximately equal to the sum over the entire dataset

$$\sum_{j=1}^{N} f(X_j) \approx \frac{N}{M} \sum_{m=1}^{M} f(X_{j_m}).$$

This approximation is more accurate for larger mini-batches.

Stochastic Variational Inference

In Stochastic Variational Inference, we doubly approximate the gradient of the ELBO:

For $\boldsymbol{\eta}_i \sim \text{MvNormal}(\mathbf{0}, I)$, $j_m \sim \text{DiscreteUniform}(1, \ldots, N)$

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}} \text{ELBO}(\phi) &\approx \frac{1}{L} \sum_{i=1}^{L} \nabla_{\boldsymbol{\mu}} \log P(\exp(\boldsymbol{\omega}) \cdot \boldsymbol{\eta}_i + \boldsymbol{\mu}, X) \\
&\approx \frac{N}{M} \frac{1}{L} \sum_{m=1}^{M} \sum_{i=1}^{L} \nabla_{\boldsymbol{\mu}} \log P(\exp(\boldsymbol{\omega}) \cdot \boldsymbol{\eta}_i + \boldsymbol{\mu}, X_{j_m})
\end{aligned}$$

Typically, for $M \approx 100$ large enough, L can be set to 1.

The complexity of SVI is $\mathcal{O}(MLK)$, where $M \ll N$ is the mini-batch size, L is the number of samples per iteration, and K is the number of variables.

Resources

KL Divergence - Clearly explained!
https://www.youtube.com/watch?v=9_eZHt2qJs4

Variational Inference + ELBO Intuition
https://www.youtube.com/watch?v=HxQ94L8n0vU

Automatic Differentiation and Gradient Descent
https://medium.com/@rhome/automatic-differentiation-26d5a993692b

Paper: Automatic Differentiation Variational Inference
https://arxiv.org/pdf/1603.00788.pdf

Paper: Stochastic Variational Inference
https://arxiv.org/pdf/1312.6114.pdf