# Probabilistic Programming

## Modern tools for probabilistic modeling and inference

Christian Findenig, Thomas Wedenig

Graz University of Technology, Austria
Institute of Theoretical Computer Science

# Table of contents

# Motivation

### Example

- Suppose you have a **coin**

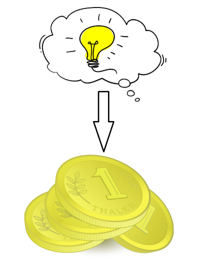

- You are unsure about its **fairness**
  - $z \in [0, 1]$
- You collect **data** by tossing it
  - $\mathbf{x} = [1, 0, 0, \ldots, 1]$

### Frequentist Approach

- **Maximum Likelihood!**
- Likelihood $p(x_i \mid z) = \text{Bernoulli}(z)$
- Tosses are i.i.d.:

$$p(\mathbf{x} \mid z) = \prod_{i=1}^{n} p(x_i \mid z)$$

- Find $z^*$ that maximizes $p(\mathbf{x} \mid z)$
  - $z^* = \texttt{np.mean(x)}$

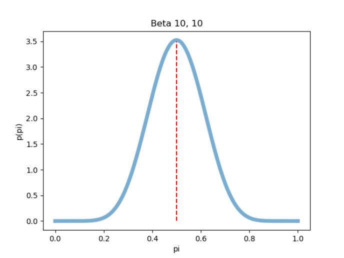


## Prior

$$p(z) = \mathrm{Beta}(10, 10)$$

## Likelihood

$$p(\mathbf{x} \mid z) = \prod_{i=1}^{n} p(x_i \mid z)$$




## Posterior

$$p(z \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid z)p(z)}{p(\mathbf{x})} = \frac{p(\mathbf{x} \mid z)p(z)}{\int p(\mathbf{x} \mid z')p(z')\, dz'}$$

How do we compute this?

## Let's do the math!

$$
\begin{aligned}
p(z \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid z)p(z)}{\int p(\mathbf{x} \mid z')p(z')\,dz'} \\
&= \frac{\prod_{i=1}^{n} z^{x_i}(1-z)^{1-x_i}\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} z^{a-1}(1-z)^{b-1}}{\int \prod_{i=1}^{n} z'^{x_i}(1-z')^{1-x_i}\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} z'^{a-1}(1-z')^{b-1}\,dz'} \\
&= \frac{\prod_{i=1}^{n} z^{x_i}(1-z)^{1-x_i} z^{a-1}(1-z)^{b-1}}{\int \prod_{i=1}^{n} z'^{x_i}(1-z')^{1-x_i} z'^{a-1}(1-z')^{b-1}\,dz'} \\
&= \frac{\prod_{i=1}^{n} z^{x_i}(1-z)^{1-x_i} z^{a-1}(1-z)^{b-1}}{\Gamma(a+\sum_{i=1}^{n} x_i)\Gamma(b+\sum_{i=1}^{n}(1-x_i))\Gamma^{-1}(a+b+n)} \\
&= \mathrm{Beta}\left(a + \sum_{i=1}^{n} x_i,\ b + n - \sum_{i=1}^{n} x_i\right)
\end{aligned}
$$

- We were able to **analytically** solve the integral $p(\mathbf{x})$!
- Does this work for *any* choice for prior and likelihood?
  - Unfortunately, no! 😩

**In general, we're not so lucky**

- Modeling should be **flexible**
  - $\implies$ **inference becomes difficult**
- Let's use Probabilistic Programming

# Abstraction

## Abstraction

Computer Science is all about abstraction.

- People want to **specify a model**
  - e.g., prior and likelihood
- Inference is done **by a tool**

## (Informal) Definition

Probabilistic programming is about
doing statistics using the tools of computer science.

**Tooling**

Tooling is important.

**Hypothesis**

Deep Learning Revolution would have been impossible without auto-diff.

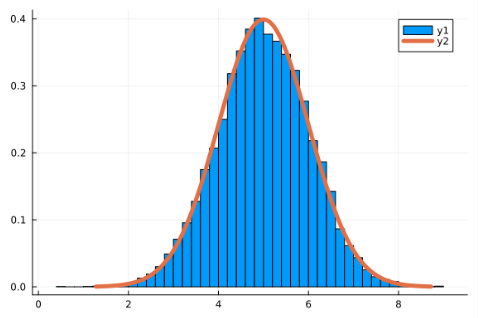
- Flexible and rich **model specification**
- Should encourage **prototyping** and **iterative model development**
- $\Rightarrow$ Let's use a **programming language**!

```python
import pymc as pm

model = pm.Model()
x = [1, 1, 1, 0, 1, 1, 1, 1, 0, 1]
with model:
    # Prior p(z)
    z = pm.Beta("z", alpha=10, beta=10)

    # Likelihood p(x | z)
    x_likelihood = pm.Bernoulli("x_likelihood", p=z, observed=x)
```

- Build on top of **auto-diff frameworks**
  - PyTorch, JAX, Tensorflow, . . .
- Support for
  - Deep Generative Modeling
  - Gaussian/Dirichlet Processes
  - Discrete Latent Variables
  - . . .

- Markov Chain Monte Carlo
  - Generate samples from the true posterior
- Variational Inference
  - Approximate the posterior with a tractable distribution

# Markov-Chain Monte Carlo




Metropolis Hasting

## Prior, Likelihood & Posterior

$$p(z) = \mathrm{Beta}(10, 10)$$

$$p(\mathbf{x} \mid z) = \prod_{i=1}^{n} p(x_i \mid z)$$

$$p(z \mid \mathbf{x}) \propto p(\mathbf{x} \mid z) \cdot p(z)$$

## The algorithm

1. Propose $z$

$$z_{i+1} \sim g(z_{i+1} \mid z_i)$$

2. Ratio

$$R = \frac{p(z_{i+1} \mid \mathbf{x})}{p(z_i \mid \mathbf{x})}$$

3. $z_{i+1}$ with $Bernoulli(min(R, 1))$, else $z_i$

## Ratio within posterior

$$R = \frac{p(z_{i+1} \mid \mathbf{x})}{p(z_i \mid \mathbf{x})} = \frac{p(\mathbf{x} \mid z_{i+1}) \cdot p(z_{i+1})}{p(\mathbf{x} \mid z_i) \cdot p(z_i)}$$

## Prior, Likelihood & Posterior

$$p(z) = \mathrm{Beta}(10, 10)$$

$$p(\mathbf{x} \mid z) = \prod_{i=1}^{n} p(x_i \mid z)$$

$$p(z \mid \mathbf{x}) \propto p(\mathbf{x} \mid z) \cdot p(z)$$

### Rejection rate

- Hamiltonian MC & NUTS sampling
- \- Gradient needed
- \+ No rejection

### Metropolis Hasting

1. Propose $z$
2. Calculate acceptance Ratio R
3. Accept?

# MCMC - Demo

# Variational Inference

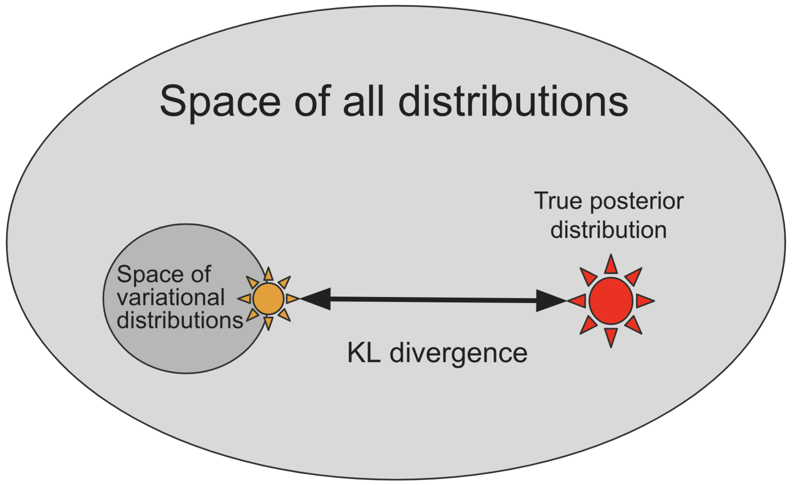
## Problem

- Computing the posterior is intractable

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p(z)}{\int p_\theta(x \mid z')p(z')\, dz'}$$

## Variational Inference

- Consider a fixed $p_\theta(x \mid z)$
- Pick a *variational* distribution $q_\phi(z)$
- Find $\phi^*$ such that $q_{\phi^*}(z)$ is close to $p_\theta(z \mid x)$




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

- When fitting $q_\phi$, we would like to minimize $D_{KL}(q_\phi(z) \mid\mid p_\theta(z \mid x))$
- But we can't compute the posterior ...
- Recall that

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_{q_\phi(z)}\left[\log p_\theta(x, z) - \log q_\phi(z)\right] \\ &= \ldots \\ &= \underbrace{\log p_\theta(x)}_{\text{constant w.r.t. } \phi} - D_{KL}(q_\phi(z) \mid\mid p_\theta(z \mid x)) \end{aligned}$$

Maximizing the **ELBO** $\Leftrightarrow$ Minimizing $D_{KL}$ to the true posterior

**ELBO**

$$\text{ELBO} = \log p_\theta(\mathrm{x}) - D_{KL}(q_\phi(\mathrm{z}) \,||\, p_\theta(\mathrm{z} \mid \mathrm{x}))$$

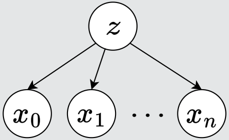
- Fit $p_\theta(\mathrm{x}, \mathrm{z})$ using maximum likelihood learning
- $\Rightarrow$ We wish to maximize $p_\theta(\mathrm{x})$

Maximizing the **ELBO** encourages this

- We optimize $p_\theta$ and $q_\phi$ simultaneously
- $\Rightarrow$ Compute $\nabla_{\theta,\phi}$ ELBO

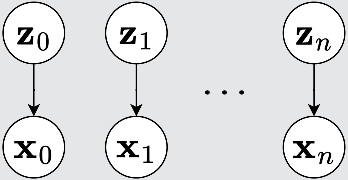
# Variational Inference Demo

## Coin Example




- One latent variable explains all coin flips
- We find $q_\phi(z \mid x_0, \ldots, x_n)$

## Image Example




- $\mathrm{z}_i$ are latent explanations of an **image** $\mathrm{x}_i$
- Find posterior estimate $q_\phi(\mathrm{z}_i \mid \mathrm{x}_i)$ **for each image**

## Classical VI

$$\mathbf{x}_0 \longrightarrow \boxed{\mathrm{VI}(p(\mathbf{x},\mathbf{z}), q_\phi(\mathbf{z}), \mathbf{x}_0)} \longrightarrow \phi^*$$

$$\phi^* = (\mu^*, \sigma^*)$$

$$q_{\phi^*}(\mathbf{z}) = \mathcal{N}(\mu^*, (\sigma^*)^2 I)$$

- $q_{\phi^*}(\mathsf{z})$ is the posterior for $\mathsf{x}_0$
- If we are given $\mathsf{x}'$, we need to re-run VI!

## Amortized VI

$$\{\mathbf{x}_0, \ldots, \mathbf{x}_n\} \longrightarrow \boxed{\mathrm{AVI}\left(p(\mathbf{x},\mathbf{z}), q_\phi(\mathbf{z} \mid \mathbf{x}), \{\mathbf{x}_0, \ldots, \mathbf{x}_n\}\right)} \longrightarrow f_{\phi^*}$$

$$f_{\phi^*}(\mathbf{x}) = (\mu^*, \sigma^*)$$

$$q_{\phi^*}(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mu^*, (\sigma^*)^2 I)$$

- $\phi^*$ refer to **NN parameters**
- **Function** from any $\mathsf{x}$ to $q_{\phi^*}(\mathsf{z} \mid \mathsf{x})$

# Practical considerations

### MCMC

- 😍 Very **general**
- 😍 Exact in the limit (infinite time)
- 😩 Computationally **very expensive**
- 😩 Convergence **hard to diagnose**

### Variational Inference

- 😍 Inference replaced by **optimization**
- 😍 More **efficient**
- 😩 How to **choose** $q_\phi(z)$?
- 😩 No **accuracy-computation tradeoff**

# Probabilistc Programming Languages

Stan

PYRO

PyMC

Edward

BUGS

infer.net

# Conclusions

Probabilistic Programming is a set of tools
for Bayesian Modeling & Inference

- **Declarative, rich modeling system**
- Abstracts away **inference routines**
- Works in tandem with techniques from **Deep Learning**

# Appendix

# ELBO Gradient Estimation

## ELBO

$$\text{ELBO} = \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})\right]$$

## ELBO Gradient Estimation w.r.t. $\theta$

$$\begin{aligned}
\nabla_\theta \, \text{ELBO} &= \nabla_\theta \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})}\left[\nabla_\theta \log p_\theta(\mathbf{x}, \mathbf{z})\right] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(\mathbf{x}, \mathbf{z}_i) \qquad \mathbf{z}_i \overset{\text{i.i.d}}{\sim} q_\phi(\mathbf{z})
\end{aligned}$$

### ELBO

$$\text{ELBO} = \mathbb{E}_{q_\phi(z)}\left[\log p_\theta(x, z) - \log q_\phi(z)\right]$$

### ELBO Gradient Estimation w.r.t. $\phi$

$$\begin{aligned}\nabla_\phi \text{ELBO} &= \nabla_\phi \mathbb{E}_{q_\phi(z)}\left[\log p_\theta(x, z) - \log q_\phi(z)\right] \\ &= ???\end{aligned}$$

- In VAEs, **we have the same problem!**
- We use the **reparametrization trick** 😎

# Reparametrization Trick

- If $q_\phi(z)$ was a 1D-Gaussian, we can produce a sample by calculating

$$z = \mu + \sigma\epsilon \qquad \epsilon \sim \mathcal{N}(0, 1)$$

- We **separate** the **sample** $\epsilon$ from the **distribution parameters** $\phi = (\mu, \sigma)^T$
- In general:

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

## ELBO Gradient Estimation w.r.t. $\phi$

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})\right] &= \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z})}\left[f(\mathbf{z})\right] \\
&= \nabla_\phi \mathbb{E}_{p(\epsilon)}\left[f(g(\epsilon, \phi))\right] \\
&= \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(g(\epsilon, \phi))\right] \\
&= \mathbb{E}_{p(\epsilon)}\left[\nabla_{\mathbf{z}} f(\mathbf{z})|_{\mathbf{z}=g(\epsilon,\phi)} \, \nabla_\phi g(\epsilon, \phi)\right]
\end{aligned}$$

# Reparametrization Trick

- Can we **reparametrize all distributions** $q_\phi$?
- **No!** 😩
- For all **discrete distributions**: $\nabla_\phi g(\epsilon, \phi)$ **does not exist**
- We can use a different estimator for $\nabla_\phi$ ELBO (REINFORCE estimator)
  - ... which suffers from **high variance**
- Can be **improved** if we **leverage dependency structure** in the model