Probabilistic Programming and AI: Lecture 2

Introduction to Probabilistic Programming Languages

Markus Böck and Jürgen Cito

Research Unit of Software Engineering

Table of contents

Recap on Bayesian Inference

Recap on Bayesian Inference

- Latent variables Θ
- Observed variables X
- Observes are believed to be generated from latents

Recap on Bayesian Inference

- Latent variables Θ
- Observed variables X
- Observes are believed to be generated from latents
- Prior distribution $P(\Theta)$ captures our belief/information about the latent variables before observing data

Recap on Bayesian Inference

- Latent variables Θ
- Observed variables X
- Observes are believed to be generated from latents
- Prior distribution $P(\Theta)$ captures our belief/information about the latent variables before observing data
- Likelihood $P(X|\Theta)$ encodes how we believe the observes are generated from latents

Recap on Bayesian Inference

- Latent variables Θ
- Observed variables X
- Observes are believed to be generated from latents
- Prior distribution $P(\Theta)$ captures our belief/information about the latent variables before observing data
- Likelihood $P(X|\Theta)$ encodes how we believe the observes are generated from latents
- Posterior distribution is obtained with Bayes Theorem

$$P(\Theta|X) = \frac{P(X|\Theta) \times P(\Theta)}{P(X)}$$

Recap on Bayesian Inference

- Latent variables Θ
- Observed variables X
- Observes are believed to be generated from latents
- Prior distribution $P(\Theta)$ captures our belief/information about the latent variables before observing data
- Likelihood $P(X|\Theta)$ encodes how we believe the observes are generated from latents
- Posterior distribution is obtained with Bayes Theorem

$$P(\Theta|X) = \frac{P(X|\Theta) \times P(\Theta)}{P(X)}$$

- Posterior is the updated belief, our information about the latents after having observed data

Recap on Bayesian Inference

Prior $P(\Theta)$

- design decision: can be constructed from expert knowledge / previous experiments
- computable

Recap on Bayesian Inference

Prior $P(\Theta)$

- design decision: can be constructed from expert knowledge / previous experiments
- **computable**

Likelihood $P(X|\Theta)$

- encodes generative process from latents Θ to observes X
- **computable**

Recap on Bayesian Inference

Prior $P(\Theta)$

- design decision: can be constructed from expert knowledge / previous experiments
- **computable**

Likelihood $P(X|\Theta)$

- encodes generative process from latents Θ to observes X
- **computable**

Joint Probability $P(X, \Theta)$

- product of prior and likelihood $P(X, \Theta) = P(X|\Theta) \times P(\Theta)$
- **computable**

Recap on Bayesian Inference

Marginal Likelihood / Model Evidence $P(X)$

- can be interpreted as the probability of generating the observed data for all possible values of the latent variables, $P(X) = \int P(X, \Theta) d\Theta$,
- it can be understood as the probability of the model itself
- can only be **approximated** in most models

Recap on Bayesian Inference

Marginal Likelihood / Model Evidence $P(X)$

- can be interpreted as the probability of generating the observed data for all possible values of the latent variables, $P(X) = \int P(X, \Theta) d\Theta$,
- it can be understood as the probability of the model itself
- can only be **approximated** in most models

Posterior $P(\Theta|X)$

- is the updated belief, the information about the latents after having observed data
- **what we want to know**
- can only be **approximated** in most models

Example

Coin model

$$\begin{aligned} p &\sim \text{Uniform}(0,1) && \textit{latent} \\ x_i &\sim \text{Bernoulli}(p) && \textit{observed} \end{aligned}$$

Example

Coin model

$$\begin{aligned} p &\sim \text{Uniform}(0,1) && \textit{latent} \\ x_i &\sim \text{Bernoulli}(p) && \textit{observed} \end{aligned}$$

$\Theta = p$, $X = (x_1, \ldots, x_n)$, $P(X, \Theta) = P(p)P(x_1|p) \cdots P(x_n|p)$

Example

Coin model

$$
\begin{aligned}
p &\sim \text{Uniform}(0,1) && \textit{latent} \\
x_i &\sim \text{Bernoulli}(p) && \textit{observed}
\end{aligned}
$$

$\Theta = p, X = (x_1, \ldots, x_n), P(X, \Theta) = P(p)P(x_1|p)\cdots P(x_n|p)$

$\underbrace{\log P(X,\Theta)}_{\text{"log-prob"}} = \underbrace{\log P(p)}_{\text{log-prior}} + \underbrace{\sum_{i=1}^{n} \log P(x_i|p)}_{\text{log-likelihood}}$

Example

Coin model

$$p \sim \text{Uniform}(0,1) \qquad \textit{latent}$$

$$x_i \sim \text{Bernoulli}(p) \qquad \textit{observed}$$

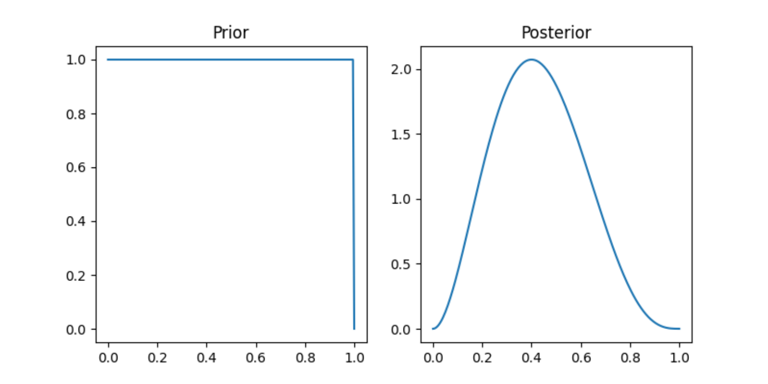
$\Theta = p, X = (x_1, \ldots, x_n), P(X, \Theta) = P(p)P(x_1|p)\cdots P(x_n|p)$

$$\underbrace{\log P(X,\Theta)}_{\text{"log-prob"}} = \underbrace{\log P(p)}_{\text{log-prior}} + \underbrace{\textstyle\sum_{i=1}^{n} \log P(x_i|p)}_{\text{log-likelihood}}$$

```python
def prior(p):
    return dist.Uniform(0,1, validate_args=False).log_prob(p).exp()

def likelihood(xs,p):
    lp = torch.tensor(0.)
    for x in xs:
        lp += dist.Bernoulli(p, validate_args=False).log_prob(x)
    return lp.exp()

def joint(xs, p):
    return prior(p) * likelihood(xs, p)

xs = torch.tensor([0.,1.,1.,0.,0.])
```

Example

```
ps = torch.linspace(0,1,200)
posterior_unnormalised = torch.hstack([joint(xs, p) for p in ps])
marginal = torch.trapz(posterior_unnormalised, ps)
posterior = posterior_unnormalised / marginal
```



Example

Compared to Frequentist Statistics

Maximum-Likelihood-Estimator:

```
ps = torch.linspace(0,1,10000)
l = torch.hstack([likelihood(xs, p) for p in ps])
ps[l.argmax()]
```

Returns:

```
0.3999
```

Compared to Frequentist Statistics

Maximum-Likelihood-Estimator:

```
ps = torch.linspace(0,1,10000)
l = torch.hstack([likelihood(xs, p) for p in ps])
ps[l.argmax()]
```

Returns:

```
0.3999
```

Hypothesis testing $\mathcal{H}_0 : p = 0.5$:

```
m = xs.mean() # test statistic: mean
means = dist.Bernoulli(0.5).sample((len(xs), 10000)).mean(dim=0)
a = min(1-m,m)
b = max(1-m,m)
# probability that sample is more extreme than observations
((means < a) | (b < means)).float().mean()
```

Returns:

```
tensor(0.3662) # p-value -> cannot reject null
```

What is conditional probability

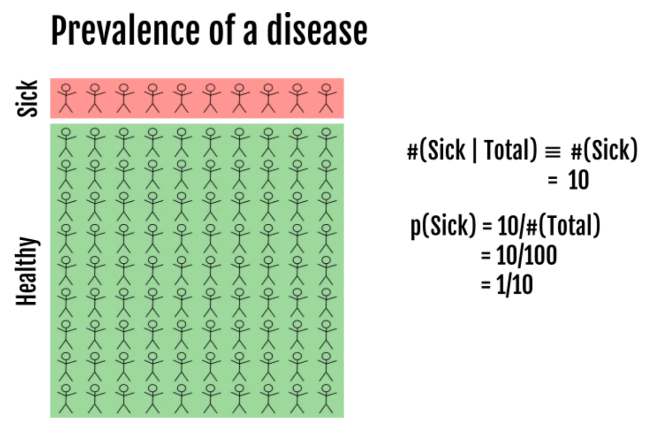
What is conditional probability

It is a ”restriction” of the sample space:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

https://seeing-theory.brown.edu/compound-probability/index.html#section3

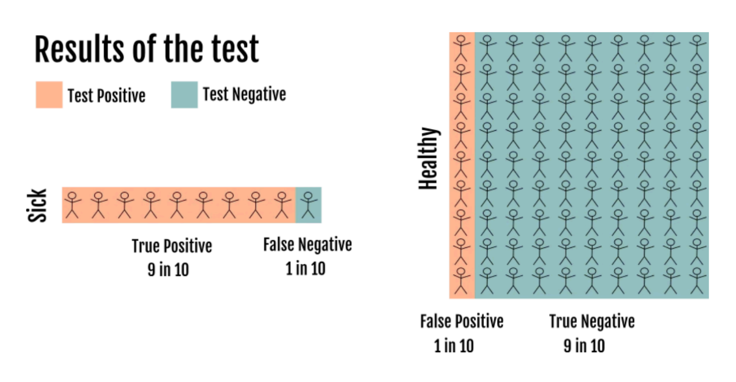
What is conditional probability



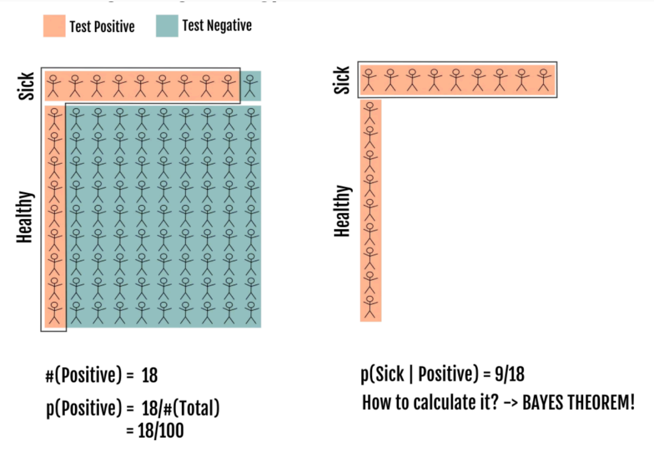
https://medium.com/@javiergb_com/

why-testing-positive-for-a-disease-may-not-mean-you-are-sick-4a3a16a290eb

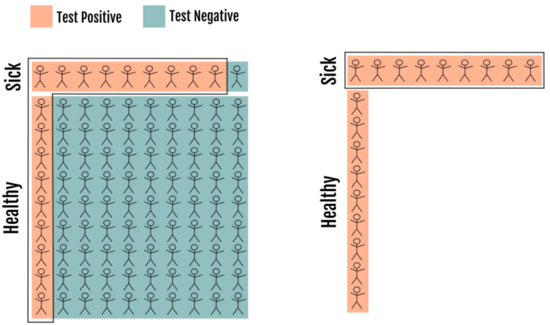
What is conditional probability



What is conditional probability



What is conditional probability



#(Positive) = 18

p(Positive) = 18/#(Total) = 18/100

p(Sick | Positive) = 9/18

How to calculate it? -> BAYES THEOREM!

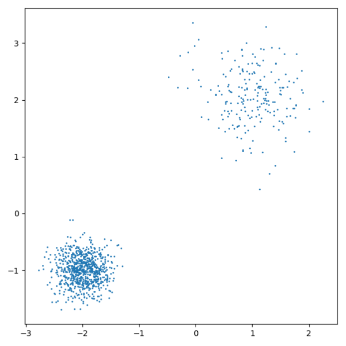
$$p(\text{Sick} \mid \text{Positive}) = \frac{p(\text{Sick}) \cdot p(\text{Positive} \mid \text{Sick})}{p(\text{Positive})}$$

Generative Modelling

Generative Modelling

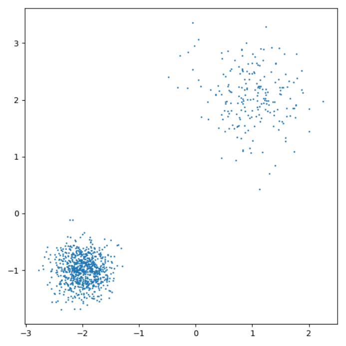
In generative modelling, we encode the way how we believe the observed data was generated in a probabilistic model.

Cluster Model



Generative Modelling: Example 1

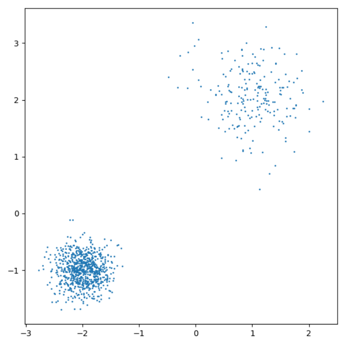
Cluster Model



- How many clusters? Easy: 2

Generative Modelling: Example 1

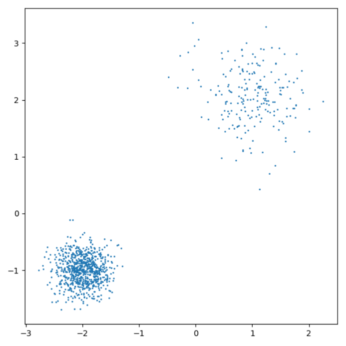
Cluster Model



- How many clusters? Easy: 2
- Model probability of being in cluster 1: $p \sim \text{Uniform}(0,1)$

Generative Modelling: Example 1

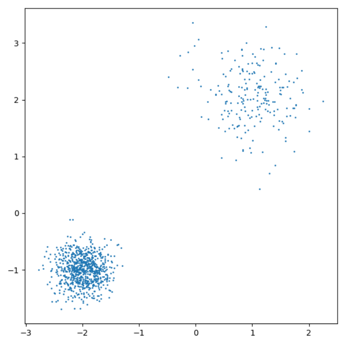
Cluster Model



- How many clusters? Easy: 2
- Model probability of being in cluster 1: $p \sim$ Uniform(0,1)
- Cluster centers, $k = 0, 1$: $\mu_x^k \sim$ Uniform(-3,3), $\mu_y^k \sim$ Uniform(-3,3)

Generative Modelling: Example 1

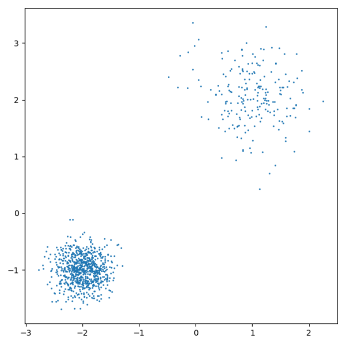
Cluster Model



- How many clusters? Easy: 2
- Model probability of being in cluster 1: $p \sim \text{Uniform}(0,1)$
- Cluster centers, $k = 0, 1$: $\mu_x^k \sim \text{Uniform}(\text{-}3,3)$, $\mu_y^k \sim \text{Uniform}(\text{-}3,3)$
- Cluster spread, $k = 0, 1$: $\sigma^k \sim \text{InverseGamma}(1,1)$

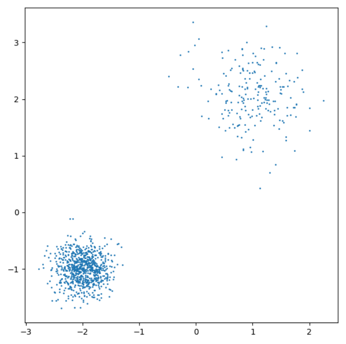
Generative Modelling: Example 1

Cluster Model



- How many clusters? Easy: 2
- Model probability of being in cluster 1: $p \sim$ Uniform(0,1)
- Cluster centers, $k = 0, 1$: $\mu_x^k \sim$ Uniform(-3,3), $\mu_y^k \sim$ Uniform(-3,3)
- Cluster spread, $k = 0, 1$: $\sigma^k \sim$ InverseGamma(1,1)
- Cluster membership, $i = 1, \ldots, N$: $z_i \sim$ Bernoulli(p)

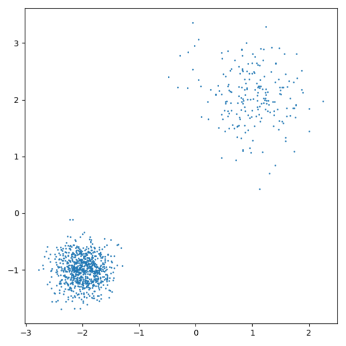
Cluster Model



- How many clusters? Easy: 2
- Model probability of being in cluster 1:
  $p \sim$ Uniform(0,1)
- Cluster centers, $k = 0, 1$:
  $\mu_x^k \sim$ Uniform(-3,3), $\mu_y^k \sim$ Uniform(-3,3)
- Cluster spread, $k = 0, 1$:
  $\sigma^k \sim$ InverseGamma(1,1)
- Cluster membership, $i = 1, \ldots, N$:
  $z_i \sim$ Bernoulli(p)
- Observed data, , $i = 1, \ldots, N$:
  $x_i \sim \mathrm{Normal}(\mu^{z_i}, \sigma^{z_i})$

Generative Modelling: Example 1

Cluster Model



- How many clusters? Easy: 2
- Model probability of being in cluster 1: $p \sim$ Uniform(0,1)
- Cluster centers, $k = 0, 1$: $\mu_x^k \sim$ Uniform(-3,3), $\mu_y^k \sim$ Uniform(-3,3)
- Cluster spread, $k = 0, 1$: $\sigma^k \sim$ InverseGamma(1,1)
- Cluster membership, $i = 1, \ldots, N$: $z_i \sim$ Bernoulli(p)
- Observed data, , $i = 1, \ldots, N$: $x_i \sim \text{Normal}(\mu^{z_i}, \sigma^{z_i})$

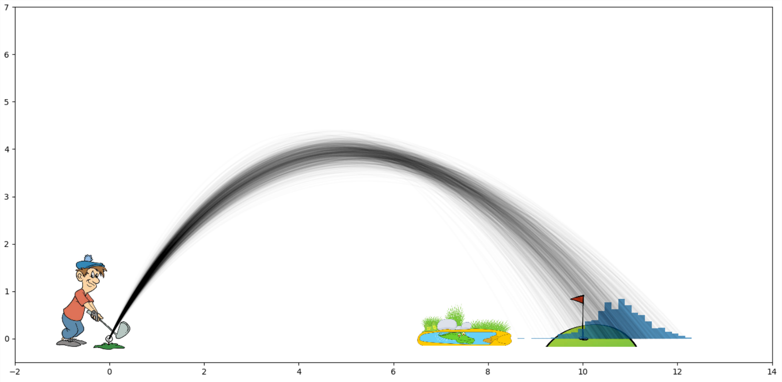
$$P(X, \Theta) = P(p)P(\mu_x^1)P(\mu_y^1)P(\mu_x^2)P(\mu_y^2)P(\sigma^1)P(\sigma^2)\prod_{i=1}^{N} P(z_i|p)P(x_i|\mu, \sigma, z_i)$$

Generative Modelling: Example 1

- How many clusters? Easy: 2
- Model probability of being in cluster 1: $p \sim$ Uniform(0,1)
- Cluster centers, $k = 0, 1$: $\mu_x^k \sim$ Uniform(-3,3), $\mu_y^k \sim$ Uniform(-3,3)
- Cluster spread, $k = 0, 1$: $\sigma^k \sim$ InverseGamma(1,1)
- Cluster membership, $i = 1, \ldots, N$: $z_i \sim$ Bernoulli(p)
- Observed data, , $i = 1, \ldots, N$: $x_i \sim \text{Normal}(\mu^{z_i}, \sigma^{z_i})$

```
@model function cluster(x::Vector{Vector{Float64}})
    K = 2 # 2 clusters
    p ~ Uniform(0.,1) # probability of being in cluster 1
    mu_x = zeros(K)
    mu_y = zeros(K)
    sigmas = zeros(K)
    for k in 1:K
        # Cluster centers
        mu_x[k] ~ Uniform(-3,3)
        mu_y[k] ~ Uniform(-3,3)
        # Cluster spread
        sigmas[k] ~ InverseGamma(1,1)
    end
    z = zeros(Int, length(x))
    for i in 1:length(x)
        z[i] ~ Bernoulli(p) # Cluster membership
        mu = z[i] == 1 ? [mu_x[1],mu_y[1]] : [mu_x[2],mu_y[2]]
        sigma = z[i] == 1 ? sigmas[1] : sigmas[2]
        x[i] ~ MvNormal(mu, sigma) # Observed data
    end
end
```

Generative Modelling: Example 2



Probabilistic Programming Languages

Probabilistic Programming Languages

Probabilistic programming is a programming paradigm in which probabilistic models are specified and posterior inference for these models is performed automatically.

Probabilistic Programming Languages

In the most general sense:

*A **probabilistic program** is a non-deterministic program for which each execution can be weighted with a real number (e.g a probability).*

Probabilistic Programming Languages

```
def linear_regression(x, y):
    lp = torch.tensor(0.0)

    slope = dist.Normal(0.,3.).sample()
    lp += dist.Normal(0.,3.).log_prob(slope)

    intercept = dist.Normal(0.,3.).sample()
    lp += dist.Normal(0.,3.).log_prob(intercept)

    sigma = dist.HalfCauchy(1).sample()
    lp += dist.HalfCauchy(1).log_prob(sigma)

    for i in range(len(x)):
        lp += dist.Normal(slope * x[i] + intercept, sigma).log_prob(y[i])


    return (slope, intercept, sigma), lp
```

Probabilistic Programming Languages

```python
def linear_regression(x, y):
    lp = torch.tensor(0.0)

    slope = dist.Normal(0.,3.).sample()
    lp += dist.Normal(0.,3.).log_prob(slope)

    intercept = dist.Normal(0.,3.).sample()
    lp += dist.Normal(0.,3.).log_prob(intercept)

    sigma = dist.HalfCauchy(1).sample()
    lp += dist.HalfCauchy(1).log_prob(sigma)

    for i in range(len(x)):
        lp += dist.Normal(slope * x[i] + intercept, sigma).log_prob(y[i])


    return (slope, intercept, sigma), lp
```

$$P(X, \Theta) = P(\text{slope})P(\text{intercept})P(\text{sigma}) \prod_{i=1}^{N} P(x_i, y_i | \text{slope}, \text{intercept}, \text{sigma})$$

Probabilistic Programming Languages

```python
def linear_regression_lp(slope, intercept, sigma, x, y):
    lp = torch.tensor(0.0)

    lp += dist.Normal(0.,3.).log_prob(slope)

    lp += dist.Normal(0.,3.).log_prob(intercept)

    lp += dist.HalfCauchy(1).log_prob(sigma)

    for i in range(len(x)):
        lp += dist.Normal(slope * x[i] + intercept, sigma).log_prob(y[i])

    return lp
```

Probabilistic Programming Languages

A **probabilistic programming language (PPL)** abstracts away the complexities of computing the program weight.

Probabilistic Programming Languages

A **probabilistic programming language (PPL)** abstracts away the complexities of computing the program weight.

A PPL allows us to conveniently specify a probabilistic model as a program (by using special syntax or primitives).

Probabilistic Programming Languages

A **probabilistic programming language (PPL)** abstracts away the complexities of computing the program weight.

A PPL allows us to conveniently specify a probabilistic model as a program (by using special syntax or primitives).

Internally, the programs are represented in a way to facilitate automated posterior inference.

Probabilistic Programming Languages

```julia
# Turing
@model function linreg(x, y)
    slope ~ Normal(0,3)
    intercept ~ Normal(0,3)
    sigma ~ InverseGamma(1,1)
    for i in eachindex(y)
        y[i] ~ Normal(slope * x[i] + intercept, sigma)
    end
end
```

Probabilistic Programming Languages

```julia
# Turing
@model function linreg(x, y)
    slope ~ Normal(0,3)
    intercept ~ Normal(0,3)
    sigma ~ InverseGamma(1,1)
    for i in eachindex(y)
        y[i] ~ Normal(slope * x[i] + intercept, sigma)
    end
end
```

```python
# Pyro
def linreg(x, y):
    slope = pyro.sample("slope", dist.Normal(0,3))
    intercept = pyro.sample("intercept", dist.Normal(0,3))
    sigma = pyro.sample("sigma", dist.HalfCauchy(1))

    for i in range(len(x)):
        pyro.sample(
            f"y[{i}]",
            dist.Normal(slope * x[i] + intercept, sigma),
            obs=y[i])
```

Representation-Based

- Translate the program to internal representation
- Usually the underlying representation is some sort of graph
- Optimised inference

Probabilistic Programming Languages

Representation-Based

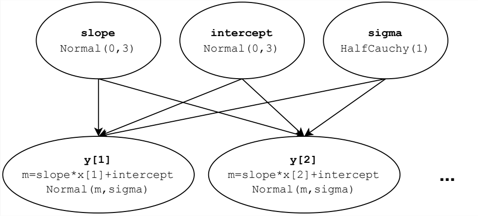
- Translate the program to internal representation
- Usually the underlying representation is some sort of graph
- Optimised inference

Evaluation-Based

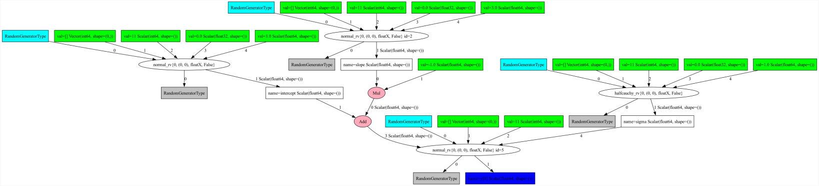
- Usually embedded in existing languages
- Have special programming constructs to specify models
- Run the entire program in different contexts for inference
- General Purpose

Example: PyMC

```
x = np.array([-1.0, -0.5, 0.0, 0.5, 1.0])
y = np.array([-1.2, -1.5, -0.0, -0.8, 1.5])
with pymc.Model() as linreg:
    slope = pymc.Normal("slope", 0.,3.)
    intercept = pymc.Normal("intercept", 0.,3.)
    sigma = pymc.HalfCauchy("sigma", 1)
    for i in range(len(y)):
        pymc.Normal(
            f"y[{i}]",
            slope * x[i] + intercept, sigma,
            observed=y[i])
```



Underlying Computation Graph in PyMC



Representation-Based: Graphical Models

Strengths

- Can exploit structure of model
- Optimised inference

Constraints

- Only works for models with finite number of random variables
- Are not ”embedded” in host language

Representation-Based: Unconstrained Function

Example: Stan

```
data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real slope;
  real intercept;
  real<lower=0> sigma;
}
model {
  slope ~ normal(0,3)
  intercept ~ normal(0,3)
  sigma ~ inv_gamma(2,1)
  y ~ normal(slope+intercept*x,sigma);
}
```

```
double model(vector<double> theta,
    int N, vector<double> x, vector<double> y) {

    double target = 0.0;

    slope = theta[0];
    intercept = theta[1];
    sigma = exp(theta[2]); // transformed

    target += normal_lpdf(slope, 0, 3);
    target += normal_lpdf(intercept, 0, 3);
    target += inv_gamma_lpdf(sigma, 2, 1);

    for (int i = 0; i++, i < N) {
        target += normal_lpdf(y[i],slope*x[i]+intercept,sigma)
    }

    return target;
}
```

$$\text{model}: \mathbb{R}^3 \to \mathbb{R}$$

a.s. differentiable if all variables are continuous and terminates

Representation-Based: Unconstrained Function

Strengths

- Specialised (gradient-based) inference
- Fast inference

Constraints

- Black box function
- Only supports finite number of random variables

Example: Psi

```
def main(){
    flips := [0,1,1,0,0];
    n := flips.length;

    p := uniform(0,1);

    for i in [0..n) {
        x := bernoulli(p);
        observe(x==flips[i]);
    }

    u := bernoulli(p);
    return u;
}
```

$$p(u) \quad = \quad \frac{\int_0^1 p^{2+u}(1-p)^{4-u}dp}{\int_0^1 p^3(1-p)^3 dp + \int_0^1 p^2(1-p)^4 dp}$$

Representation-Based: Symbolic

Example: Psi

```
def main(){
    flips := [0,1,1,0,0];
    n := flips.length;

    p := uniform(0,1);

    for i in [0..n) {
        x := bernoulli(p);
        observe(x==flips[i]);
    }

    u := bernoulli(p);
    return u;
}
```

$$
\begin{aligned}
p(u) &= \frac{\int_0^1 p^{2+u}(1-p)^{4-u}dp}{\int_0^1 p^3(1-p)^3dp + \int_0^1 p^2(1-p)^4dp} \\
&= \frac{\int_0^1 p^{2+u}(1-p)^{4-u}dp}{\frac{1}{140}+\frac{1}{105}} \\
&= \frac{3}{7}\delta_1(u) + \frac{4}{7}\delta_0(u)
\end{aligned}
$$

Representation-Based: Symbolic

Strengths

- Exact inference

Constraints

- Most models are not solvable symbolically

Evaluation-Based

Example: Pyro

```
x = torch.tensor(-1.0, -0.5, 0.0, 0.5, 1.0])
y = torch.tensor([-1.2, -1.5, -0.0, -0.8, 1.5])

def linreg(x):
    slope = pyro.sample("slope", dist.Normal(0,3))
    intercept = pyro.sample("intercept", dist.Normal(0,3))
    sigma = pyro.sample("sigma", dist.HalfCauchy(1))

    for i in range(len(x)):
        pyro.sample(
            f"y[{i}]",
            dist.Normal(slope * x[i] + intercept, sigma))

    return slope, intercept
```

```
linreg(x)
```

returns:

```
(tensor(0.4707), tensor(3.9712))
```

```
with pyro.poutine.trace() as trace:
    linreg(x)
trace.get_trace().nodes["sigma"]
```

returns:

```
{'type': 'sample',
 'name': 'sigma',
 'fn': HalfCauchy(),
 'is_observed': False,
 'value': tensor(4.4170),
 ...}
```

Example: Pyro

```
x = torch.tensor(-1.0, -0.5, 0.0, 0.5, 1.0])
y = torch.tensor([-1.2, -1.5, -0.0, -0.8, 1.5])

def linreg(x):
    slope = pyro.sample("slope", dist.Normal(0,3))
    intercept = pyro.sample("intercept", dist.Normal(0,3))
    sigma = pyro.sample("sigma", dist.HalfCauchy(1))

    for i in range(len(x)):
        pyro.sample(
            f"y[{i}]",
            dist.Normal(slope * x[i] + intercept, sigma))

    return slope, intercept
```

```
with pyro.poutine.trace() as trace:
    with pyro.condition(
        data={f"y[{i}]": y[i] for i in range(5)}):
        linreg(x)
trace.get_trace().nodes["y[0]"]
```

returns:

```
{'type': 'sample',
 'name': 'y[0]',
 'fn': Normal(loc: -7.59, scale: 2.31),
 'is_observed': True,
 'value': tensor(-1.2000),
 ...}
```

```
trace.get_trace().log_prob_sum()
```

returns:

```
tensor(-27.8861)
```

Evaluation-Based

Example: Gen

```
@gen function linreg(x, y)
    slope ~ normal(0,3)
end
```

```
function var"##linreg#369"(var"##state#294", x::Any, y::Any)
    slope = Gen.traceat(var"##state#294", normal, (0, 3), :slope)
end
```

Evaluation-Based

```
@model function linreg(x, y) # Turing
    slope ~ Normal(0,3)
end
```

```
function linreg(__model__::Model, __varinfo__::AbstractVarInfo, __context__::AbstractContext, x, y; )
    var"##dist#362" = Normal(0, 3)
    var"##vn#359" = (DynamicPPL.resolve_varnames)((AbstractPPL.VarName){:slope}(), var"##dist#362")
    var"##isassumption#360" = begin
            if (DynamicPPL.contextual_isassumption)(__context__, var"##vn#359")
                ...
            end
        end
    begin
        var"##retval#364" = if (DynamicPPL.contextual_isfixed)(__context__, var"##vn#359")
                slope = (DynamicPPL.getfixed_nested)(__context__, var"##vn#359")
            elseif var"##isassumption#360"
                begin
                    (var"##value#363", __varinfo__) = (DynamicPPL.tilde_assume!!)(__context__, (DynamicPPL.unwra
                    slope = var"##value#363"
                    var"##value#363"
                end
            else
                if !((DynamicPPL.inargnames)(var"##vn#359", __model__))
                    slope = (DynamicPPL.getconditioned_nested)(__context__, var"##vn#359")
                end
                (var"##value#361", __varinfo__) = (DynamicPPL.tilde_observe!!)(__context__, (DynamicPPL.check_ti
                var"##value#361"
            end
        return (var"##retval#364", __varinfo__)
    end
end
```

Evaluation-Based

Strengths

- Can represent any probabilistic model
- Can be embedded in host languages

Constraints

- General-purpose inference is difficult

Implementing a minimal PPL

Implementing a minimal PPL

Goals:

- Evaluation-based
- Contextualised execution *(Lecture 2)*

Implementing a minimal PPL

Goals:

- Evaluation-based
- Contextualised execution *(Lecture 2)*
- General-purpose inference:
    - Likelihood weighting *(Lecture 2 + Assignment 2)*
    - Metropolis Hastings *(Lecture 3 + Assignment 3)*

Implementing a minimal PPL

Goals:

- Evaluation-based
- Contextualised execution *(Lecture 2)*
- General-purpose inference:
  - Likelihood weighting *(Lecture 2 + Assignment 2)*
  - Metropolis Hastings *(Lecture 3 + Assignment 3)*
- Gradient-based inference:
  - Hamiltonian Monte Carlo *(Lecture 3 + Assignment 4)*
  - Variational Inference (*Lecture 4 + Assignment 4)*

torch.distributions

```
class Distribution:
    def sample(self, sample_shape: torch.Size = torch.Size()) -> torch.Tensor:
        raise NotImplementedError

    def log_prob(self, value: torch.Tensor) -> torch.Tensor:
        raise NotImplementedError
```

PPL Core: Sample Context

```python
_SAMPLE_CONTEXT = None

class SampleContext(ABC):
    def __enter__(self):
        global _SAMPLE_CONTEXT
        _SAMPLE_CONTEXT = self

    def __exit__(self, *args):
        global _SAMPLE_CONTEXT
        _SAMPLE_CONTEXT = None

    @abstractmethod
    def sample(self,
               address: str,
               distribution: dist.Distribution,
               observed: Optional[torch.Tensor] = None) -> torch.Tensor:
        raise NotImplementedError
```

PPL Core: Sample Context

```python
ctx = MySampleContext()
with ctx:
    do_something()
```

```python
ctx = MySampleContext()
ctx.__enter__() # global _SAMPLE_CONTEXT = ctx
do_something()
ctx.__exit__()  # global _SAMPLE_CONTEXT = None
```

PPL Core: Sample Statement

```python
def sample(address: str,
           distribution: dist.Distribution,
           observed: Optional[torch.Tensor] = None) -> torch.Tensor:
    global _SAMPLE_CONTEXT

    # default behavior
    if _SAMPLE_CONTEXT is None:
        if observed is not None:
            return observed
        return distribution.sample()

    # context specific behavior
    return _SAMPLE_CONTEXT.sample(address, distribution, observed)
```

Example Program

```
def noisy_geometric(p):
    x = 0
    while True:
        b = sample(f"b_{x}", dist.Bernoulli(p))
        if b:
            break
        x += 1
    y = sample("y", dist.Normal(x,1), observed=torch.tensor(3.))
    return x

```

```
torch.manual_seed(0)
[noisy_geometric(0.25) for _ in range(10)]
```

returns:

```
[2, 0, 8, 0, 4, 8, 0, 3, 1, 0]
```

Example Context

```python
class LogProb(SampleContext):
    def __init__(self, trace):
        self.log_prob = torch.tensor(0.)
        self.trace = trace

    def sample(self,
               address: str,
               distribution: dist.Distribution,
               observed: Optional[torch.Tensor] = None) -> torch.Tensor:

        if observed is not None:
            value = observed
        else:
            value = self.trace[address]

        self.log_prob += distribution.log_prob(value)

        return value
```

Example Context (cont.)

```
torch.manual_seed(0)
ctx = LogProb({
    "b_0": torch.tensor(0.),
    "b_1": torch.tensor(0.),
    "b_2": torch.tensor(1.)})
with ctx:
    x = noisy_geometric(0.25)
x, ctx.log_prob
```

returns:

```
(2, tensor(-3.3806))
```

```
p = torch.tensor(0.25)
2*torch.log(1-p) + torch.log(p) + dist.Normal(2,1).log_prob(torch.tensor(3))
# == tensor(-3.3806)
```

Example Context

```python
class Trace(SampleContext):
    def __init__(self):
        self.trace = {}

    def sample(self,
                address: str,
                distribution: dist.Distribution,
                observed: Optional[torch.Tensor] = None) -> torch.Tensor:

        if observed is not None:
            value = observed
        else:
            value = distribution.sample()

        self.trace[address] = {
            'value': value,
            'distribution': distribution,
            'is_observed': observed is not None,
            'log_prob': distribution.log_prob(value)
        }

        return value
```

Example Context (cont.)

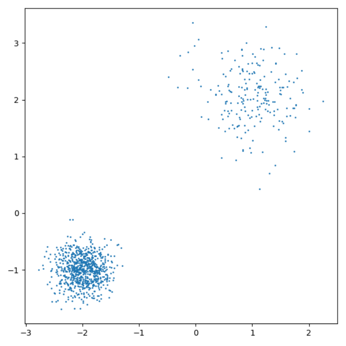
```
torch.manual_seed(0)
ctx = Trace()
with ctx:
    x = noisy_geometric(0.25)
ctx.trace
```

returns:

```
{'b_0': {'value': tensor(0.),
   'distribution': Bernoulli(probs: 0.25, logits: -1.0986123085021973),
   'is_observed': False,
   'log_prob': tensor(-0.2877)},
  'b_1': {'value': tensor(0.),
   'distribution': Bernoulli(probs: 0.25, logits: -1.0986123085021973),
   'is_observed': False,
   'log_prob': tensor(-0.2877)},
  'b_2': {'value': tensor(1.),
   'distribution': Bernoulli(probs: 0.25, logits: -1.0986123085021973),
   'is_observed': False,
   'log_prob': tensor(-1.3863)},
  'y': {'value': tensor(3),
   'distribution': Normal(loc: 2.0, scale: 1.0),
   'is_observed': True,
   'log_prob': tensor(-1.4189)}}
```

Generative Modelling: Example 1

Cluster Model



- How many clusters? Easy: 2
- Model probability of being in cluster 1:
  $p \sim \text{Uniform}(0{,}1)$
- Cluster centers, $k = 0, 1$:
  $\mu_x^k \sim \text{Uniform}(\text{-}3{,}3)$, $\mu_y^k \sim \text{Uniform}(\text{-}3{,}3)$
- Cluster spread, $k = 0, 1$:
  $\sigma^k \sim \text{InverseGamma}(1{,}1)$
- Cluster membership, $i = 1, \ldots, N$:
  $z_i \sim \text{Bernoulli(p)}$
- Observed data, , $i = 1, \ldots, N$:
  $x_i \sim \text{Normal}(\mu^{z_i}, \sigma^{z_i})$

$$P(X, \Theta) = P(p)P(\mu_x^1)P(\mu_y^1)P(\mu_x^2)P(\mu_y^2)P(\sigma^1)P(\sigma^2)\prod_{i=1}^{N} P(z_i|p)P(x_i|\mu, \sigma, z_i)$$

Example Program

```
def cluster(x):
    K = 2 # 2 clusters
    p =  sample("p", dist.Uniform(0.,1)) # probability of being in cluster 1
    mu_x = []
    mu_y = []
    sigmas = []
    for k in range(K):
        # Cluster centers
        mu_x.append(sample(f"mu_x_{k}", dist.Uniform(-3,3)))
        mu_y.append(sample(f"mu_y_{k}", dist.Uniform(-3,3)))
        # Cluster spread
        sigmas.append(sample(f"sigma_{k}", dist.HalfCauchy(1)))
    for i in range(len(x)):
        z = sample(f"z_{i}", dist.Bernoulli(p)) # Cluster membership
        mu = torch.hstack([mu_x[0],mu_y[0]]) if z == 1  else torch.hstack([mu_x[1],mu_y[1]])
        sigma = sigmas[0] if z == 1 else sigmas[1]
        sample(f"x_{i}", dist.Normal(mu, sigma), observed=x[i]) # Observed data
```

First General-Purpose Inference

Rejection Sampling

Coin model:

$$p \sim \text{Uniform}(0, 1)$$
$$x_i \sim \text{Bernoulli}(p)$$

```python
def rejection_sampling(xs_observed):
    while True:
        # generate samples from joint
        p = dist.Uniform(0,1).sample()
        xs = dist.Bernoulli(p).sample(xs_observed.shape)
        if (xs == xs_observed).all():
            return p # accept
        # reject
```

Rejection Sampling

Coin model:

$$p \sim \text{Uniform}(0, 1)$$
$$x_i \sim \text{Bernoulli}(p)$$

```python
def rejection_sampling(xs_observed):
    while True:
        # generate samples from joint
        p = dist.Uniform(0,1).sample()
        xs = dist.Bernoulli(p).sample(xs_observed.shape)
        if (xs == xs_observed).all():
            return p # accept
        # reject
```

```
Rejected 50.54% of iterations for 1 number of observations.
Rejected 83.78% of iterations for 2 number of observations.
Rejected 91.76% of iterations for 3 number of observations.
Rejected 96.68% of iterations for 4 number of observations.
Rejected 98.26% of iterations for 5 number of observations.
```

First General-Purpose Inference

Likelihood Weighting (Importance Sampling)

Problem: We want to compute statistic $r(\Theta)$ for the posterior $P(\Theta|X)$, but we cannot sample from it or evaluate the density directly.

Solution: Sample from a reference distribution Q.

Likelihood Weighting (Importance Sampling)

Problem: We want to compute statistic $r(\Theta)$ for the posterior $P(\Theta|X)$, but we cannot sample from it or evaluate the density directly.

Solution: Sample from a reference distribution Q.

$$\mathbb{E}_{\Theta\sim P(.|X)}\left[r(\Theta)\right]$$

Posterior mean: $r(\Theta) = \Theta$, probability that $\Theta > 0$: $r(\Theta) = \delta_{\Theta>0}$.

First General-Purpose Inference

Importance Sampling

$$\mathbb{E}_{\Theta \sim P(.|X)} \left[r(\Theta) \right]$$

First General-Purpose Inference

Importance Sampling

$$\mathbb{E}_{\Theta \sim P(.|X)}\left[r(\Theta)\right] \quad = \quad \int r(\theta) p_{\Theta|X}(\theta|X) d\theta$$

Importance Sampling

$$\begin{aligned}\mathbb{E}_{\Theta\sim P(.|X)}\left[r(\Theta)\right] &= \int r(\theta)p_{\Theta|X}(\theta|X)d\theta \\ &= \int r(\theta)\frac{p_{\Theta|X}(\theta|X)}{p_Q(\theta)}p_Q(\theta)d\theta\end{aligned}$$

Importance Sampling

$$\begin{aligned}
\mathbb{E}_{\Theta \sim P(.|X)}\left[r(\Theta)\right] &= \int r(\theta) p_{\Theta|X}(\theta|X) d\theta \\
&= \int r(\theta) \frac{p_{\Theta|X}(\theta|X)}{p_Q(\theta)} p_Q(\theta) d\theta \\
&= \mathbb{E}_{\Theta \sim Q}\left[r(\Theta) \frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right]
\end{aligned}$$

Importance Sampling

$$\begin{aligned}\mathbb{E}_{\Theta\sim P(.|X)}\left[r(\Theta)\right] &= \int r(\theta)p_{\Theta|X}(\theta|X)d\theta \\ &= \int r(\theta)\frac{p_{\Theta|X}(\theta|X)}{p_Q(\theta)}p_Q(\theta)d\theta \\ &= \mathbb{E}_{\Theta\sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right]\end{aligned}$$

We can use samples θ_i from Q in Monte Carlo simulation:

$$\mathbb{E}_{X\sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \approx \frac{1}{N}\sum_{i=1}^{N} r(\theta_i)\frac{p_{\Theta|X}(\theta_i|X)}{p_Q(\theta_i)}$$

Importance Sampling

$$
\begin{aligned}
\mathbb{E}_{\Theta\sim P(.|X)}\left[r(\Theta)\right] &= \int r(\theta)p_{\Theta|X}(\theta|X)d\theta \\
&= \int r(\theta)\frac{p_{\Theta|X}(\theta|X)}{p_Q(\theta)}p_Q(\theta)d\theta \\
&= \mathbb{E}_{\Theta\sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right]
\end{aligned}
$$

We can use samples θ_i from Q in Monte Carlo simulation:

$$
\mathbb{E}_{X\sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \approx \frac{1}{N}\sum_{i=1}^{N} r(\theta_i)\frac{p_{\Theta|X}(\theta_i|X)}{p_Q(\theta_i)}
$$

But we cannot compute $p_{\Theta|X}(\theta_i|X)$!?

But we cannot compute $p_{\Theta|X}(\theta_i|X)$!?

$$\frac{p_{\Theta|X}(\theta_i|X)}{p_Q(\theta_i)}$$

First General-Purpose Inference

But we cannot compute $p_{\Theta|X}(\theta_i|X)$!?

$$\frac{p_{\Theta|X}(\theta_i|X)}{p_Q(\theta_i)} \quad = \quad \frac{p_{\Theta,X}(\theta_i, X)}{p_X(X)p_Q(\theta_i)}$$

First General-Purpose Inference

But we cannot compute $p_{\Theta|X}(\theta_i|X)$!?

$$\begin{aligned}\frac{p_{\Theta|X}(\theta_i|X)}{p_Q(\theta_i)} &= \frac{p_{\Theta,X}(\theta_i,X)}{p_X(X)p_Q(\theta_i)} \\ &= \frac{1}{p_X(X)}\underbrace{\frac{p_{\Theta,X}(\theta_i,X)}{p_Q(\theta_i)}}_{w_i:=}\end{aligned}$$

But we cannot compute $p_X(X)$!?

First General-Purpose Inference

But we cannot compute $p_X(X)$!?

Set $r(\Theta) = 1$

$$1 = \mathbb{E}_{X \sim P(.|X)} [r(\Theta)]$$

First General-Purpose Inference

But we cannot compute $p_X(X)$!?

Set $r(\Theta) = 1$

$$1 = \mathbb{E}_{X \sim P(.|X)}\left[r(\Theta)\right] \quad = \quad \mathbb{E}_{\Theta \sim Q}\left[\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right]$$

First General-Purpose Inference

But we cannot compute $p_X(X)$!?

Set $r(\Theta) = 1$

$$
\begin{aligned}
1 = \mathbb{E}_{X \sim P(.|X)}\left[r(\Theta)\right] &= \mathbb{E}_{\Theta \sim Q}\left[\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \\
&= \frac{1}{p_X(X)} \mathbb{E}_{\Theta \sim Q}\left[\frac{p_{\Theta,X}(\Theta, X)}{p_Q(\Theta)}\right]
\end{aligned}
$$

First General-Purpose Inference

But we cannot compute $p_X(X)$!?

Set $r(\Theta) = 1$

$$
\begin{aligned}
1 = \mathbb{E}_{X \sim P(.|X)}\left[r(\Theta)\right] \quad &= \quad \mathbb{E}_{\Theta \sim Q}\left[\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \\
&= \quad \frac{1}{p_X(X)}\mathbb{E}_{\Theta \sim Q}\left[\frac{p_{\Theta,X}(\Theta, X)}{p_Q(\Theta)}\right] \approx \frac{1}{p_X(X)} \frac{1}{N} \sum_{i=1}^{N} W_i
\end{aligned}
$$

First General-Purpose Inference

But we cannot compute $p_X(X)$!?

Set $r(\Theta) = 1$

$$
\begin{aligned}
1 = \mathbb{E}_{X\sim P(.|X)}\left[r(\Theta)\right] &= \mathbb{E}_{\Theta\sim Q}\left[\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \\
&= \frac{1}{p_X(X)}\mathbb{E}_{\Theta\sim Q}\left[\frac{p_{\Theta,X}(\Theta,X)}{p_Q(\Theta)}\right] \approx \frac{1}{p_X(X)}\frac{1}{N}\sum_{i=1}^{N} W_i
\end{aligned}
$$

Thus,

$$
p_X(X) \approx \frac{1}{N}\sum_{i=1}^{N} W_i
$$

First General-Purpose Inference

Putting it all together: $W_i = \frac{p_{\Theta,X}(\theta_i, X)}{p_Q(\theta_i)}$

$$\mathbb{E}_{\Theta \sim P(.|X)}\left[r(\Theta)\right]$$

First General-Purpose Inference

Putting it all together: $W_i = \frac{p_{\Theta,X}(\theta_i, X)}{p_Q(\theta_i)}$

$$\mathbb{E}_{\Theta\sim P(.|X)}\left[r(\Theta)\right] \quad = \quad \mathbb{E}_{\Theta\sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right]$$

First General-Purpose Inference

Putting it all together: $W_i = \frac{p_{\Theta,X}(\theta_i, X)}{p_Q(\theta_i)}$

$$\begin{aligned}
\mathbb{E}_{\Theta \sim P(.|X)}\left[r(\Theta)\right] &= \mathbb{E}_{\Theta \sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\frac{p_{\Theta,X}(\theta_i, X)}{p_X(X)p_Q(\theta_i)} \cdot r(\theta_i)
\end{aligned}$$

First General-Purpose Inference

Putting it all together: $W_i = \frac{p_{\Theta,X}(\theta_i,X)}{p_Q(\theta_i)}$

$$
\begin{aligned}
\mathbb{E}_{\Theta\sim P(.|X)}\left[r(\Theta)\right] &= \mathbb{E}_{\Theta\sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\frac{p_{\Theta,X}(\theta_i,X)}{p_X(X)p_Q(\theta_i)}\cdot r(\theta_i) \\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{W_i}{p_X(X)}r(\theta_i)
\end{aligned}
$$

First General-Purpose Inference

Putting it all together: $W_i = \frac{p_{\Theta,X}(\theta_i,X)}{p_Q(\theta_i)}$

$$\begin{aligned}
\mathbb{E}_{\Theta\sim P(.|X)}\left[r(\Theta)\right] &= \mathbb{E}_{\Theta\sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\frac{p_{\Theta,X}(\theta_i,X)}{p_X(X)p_Q(\theta_i)}\cdot r(\theta_i) \\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{W_i}{p_X(X)}r(\theta_i) \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\frac{W_i}{\sum_{k=1}^{N}W_k}r(\theta_i), \quad \theta_i\sim Q
\end{aligned}$$

First General-Purpose Inference

Putting it all together: $W_i = \frac{p_{\Theta,X}(\theta_i,X)}{p_Q(\theta_i)}$

$$\begin{aligned}
\mathbb{E}_{\Theta\sim P(.|X)}\left[r(\Theta)\right] &= \mathbb{E}_{\Theta\sim Q}\left[r(\Theta)\frac{p_{\Theta|X}(\Theta|X)}{p_Q(\Theta)}\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\frac{p_{\Theta,X}(\theta_i,X)}{p_X(X)p_Q(\theta_i)}\cdot r(\theta_i) \\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{W_i}{p_X(X)}r(\theta_i) \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\frac{W_i}{\sum_{k=1}^{N}W_k}r(\theta_i), \quad \theta_i\sim Q
\end{aligned}$$

Q has to satisfy: $p_Q(\theta)=0 \implies p_{\Theta,X}(\theta,X)=0.$

First General-Purpose Inference

Likelihood Weighting is a special case of importance sampling where the reference distribution is the prior $Q = P_\Theta$.

The weights reduce to the likelihood

$$W_i = \frac{p_{\Theta,X}(\theta_i, X)}{p_Q(\theta_i)} = \frac{p_{\Theta,X}(\theta_i, X)}{p_\Theta(\theta_i)} = p_{X|\Theta}(X|\theta_i)$$

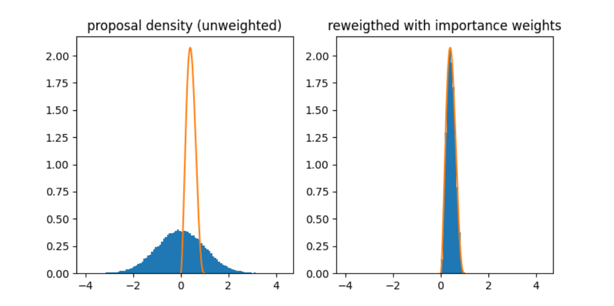
Example

Coin model

```
def prior(p):
    return dist.Uniform(0,1, validate_args=False).log_prob(p).exp()

def likelihood(xs,p):
    lp = torch.tensor(0.)
    for x in xs:
        lp += dist.Bernoulli(p, validate_args=False).log_prob(x)
    return lp.exp()

def joint(xs, p):
    return prior(p) * likelihood(xs, p)

xs = torch.tensor([0.,1.,1.,0.,0.])
```

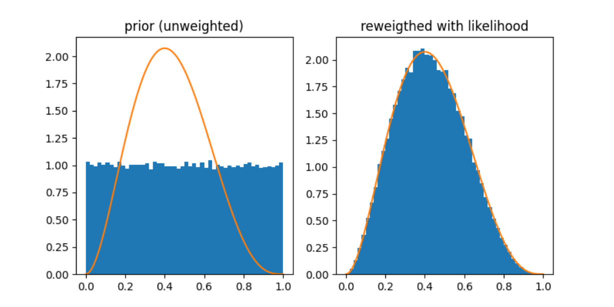
Example

```
proposal = dist.Normal(0,1)
proposal_sample = proposal.sample((N,))

weights = (
    torch.hstack([joint(xs, proposal_sample[i]) for i in range(N)]) /
    proposal.log_prob(proposal_sample).exp()
) # importance sampling

axs[0].hist(proposal_sample, density=True, bins=100)
axs[1].hist(proposal_sample, density=True, bins=100, weights=weights)
```



Example

```
proposal = dist.Uniform(0,1) # prior
proposal_sample = proposal.sample((N,))

weights = torch.hstack([likelihood(xs, proposal_sample[i]) for i in range(N)])
# likelihood weighting

axs[0].hist(proposal_sample, density=True, bins=100)
axs[1].hist(proposal_sample, density=True, bins=100, weights=weights)
```



Example

In our PPL, the coin model can be specified much more compactly.

```
def coin_model(xs):
    p = sample("p", dist.Uniform(0,1))
    for i in range(len(xs)):
        sample(f"x[{i}]", dist.Bernoulli(p), observed=xs[i])
```

In assignment 2, we will write a sample context to make likelihood weighting work automatically with any model.

Resources

Bayesian Inference Framework
https://www.youtube.com/watch?v=0w_4QcvBYII

Intuition behind Bayesian inference
https://www.youtube.com/watch?v=yvWlpwnT1nw

Bayesian posterior sampling
https://www.youtube.com/watch?v=EHqU9LE9tg8

Generative Models
http://probmods.org/chapters/generative-models.html

AI That Understands the World, Using Probabilistic Programming
https://www.youtube.com/watch?v=8j2S7BRRWus

A Personal Viewpoint on Probabilistic Programming
https://www.youtube.com/watch?v=TFXcVlKqPlM