

Probabilistic Programming and AI: Lecture 2

Introduction to Probabilistic Programming Languages

Markus Böck and Jürgen Cito

Research Unit of Software Engineering

Table of contents

1. Recap on Bayesian Inference
2. Generative Modelling
3. Probabilistic Programming Languages
4. Implementing a minimal PPL
5. First General-Purpose Inference

Recap on Bayesian Inference

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**

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 / Evidence $P(X)$

- can be interpreted as the probability of generating the observed data from a prior
- 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

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

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

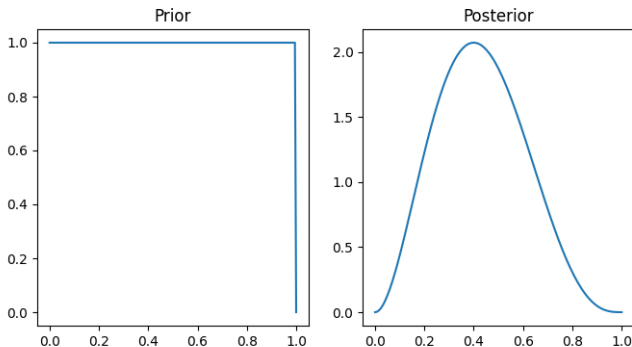
$\Theta = p, X = (x_1, \dots, x_n), P(X, \Theta) = P(p)P(x_1|p) \cdots P(x_n|p)$

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

```
1 def prior(p):
2     return dist.Uniform(0,1, validate_args=False).log_prob(p).exp()
3
4 def likelihood(xs, p):
5     lp = torch.tensor(0.)
6     for x in xs:
7         lp += dist.Bernoulli(p, validate_args=False).log_prob(x)
8     return lp.exp()
9
10 def joint(xs, p):
11     return prior(p) * likelihood(xs, p)
12
13 xs = torch.tensor([0., 1., 1., 0., 0.]
```

Example

```
1 ps = torch.linspace(0,1,200)
2 posterior_unnormalised = torch.hstack([joint(xs, p) for p in ps])
3 marginal = torch.trapz(posterior_unnormalised, ps)
4 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

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

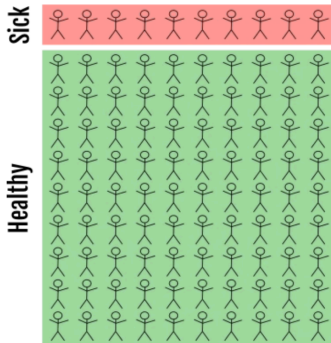
It is a "restriction" of the sample space:

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

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

What is conditional probability

Prevalence of a disease



$$\#(\text{Sick} \mid \text{Total}) \equiv \#(\text{Sick}) \\ = 10$$

$$p(\text{Sick}) = 10/\#(\text{Total}) \\ = 10/100 \\ = 1/10$$

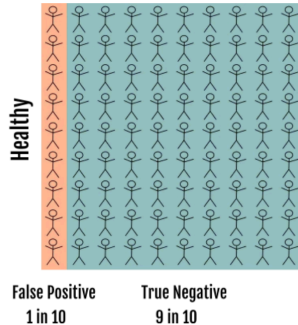
https://medium.com/@javiergb_com/

[why-testing-positive-for-a-disease-may-not-mean-you-are-sick-4a3a16a290eb](#)

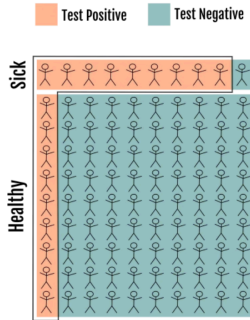
What is conditional probability

Results of the test

Test Positive Test Negative

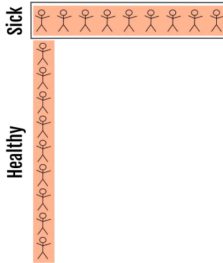


What is conditional probability



#(Positive) = 18

$p(\text{Positive}) = 18/\#(\text{Total})$
 $= 18/100$



$p(\text{Sick} | \text{Positive}) = 9/18$

How to calculate it? → BAYES THEOREM!

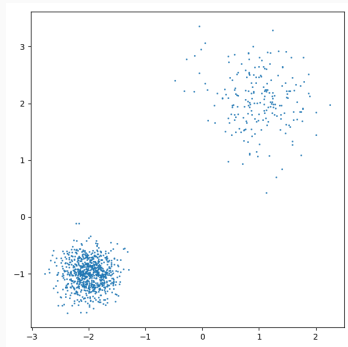
$$p(\text{Sick} | \text{Positive}) = \frac{p(\text{Sick}) \cdot p(\text{Positive} | \text{Sick})}{p(\text{Positive})}$$

Generative Modelling

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

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, $k = 0, 1$:
 $\mu_x^k \sim \text{Uniform}(-3,3)$, $\mu_y^k \sim \text{Uniform}(-3,3)$
- Cluster spread, $k = 0, 1$:
 $\sigma^k \sim \text{InverseGamma}(1,1)$
- Cluster membership, $i = 1, \dots, N$:
 $z_i \sim \text{Bernoulli}(p)$
- Observed data, $i = 1, \dots, N$:
 $x_i \sim \text{Normal}(\mu^{z_i}, \sigma^{z_i})$

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

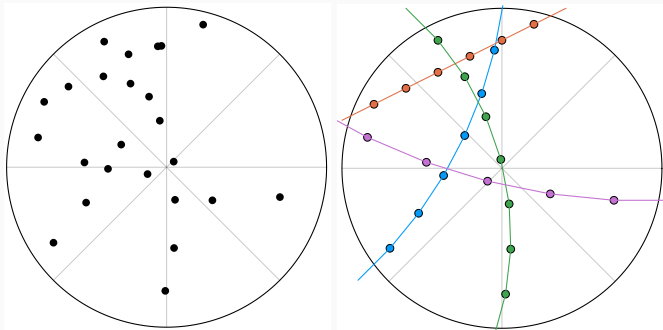
Generative Modelling: Example 1

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

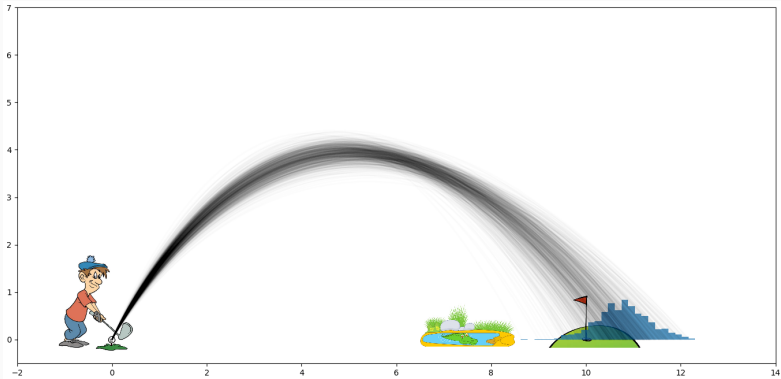
```
@model function cluster(x::Vector{Vector{Float64}})
    K = 2 # 2 clusters
    p ~ Uniform(0.,1) # probability of being in cluster 1
    mu_x = zeros(K)
    mu_y = zeros(K)
    sigmas = zeros(K)
    for k in 1:K
        # Cluster centers
        mu_x[k] ~ Uniform(-3,3)
        mu_y[k] ~ Uniform(-3,3)
        # 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

Radar Observations



Generative Modelling: Example 3



Probabilistic Programming Languages

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

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

```
1 def linear_regression(x, y):
2     lp = torch.tensor(0.0)
3
4     slope = dist.Normal(0., 3.).sample()
5     lp += dist.Normal(0., 3.).log_prob(slope)
6
7     intercept = dist.Normal(0., 3.).sample()
8     lp += dist.Normal(0., 3.).log_prob(intercept)
9
10    sigma = dist.HalfCauchy(1).sample()
11    lp += dist.HalfCauchy(1).log_prob(sigma)
12
13    for i in range(len(x)):
14        lp += dist.Normal(slope * x[i] + intercept, sigma).log_prob(y[i])
15
16
17    return (slope, intercept, sigma), lp
```

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

Probabilistic Programming Languages

```
1 def linear_regression_lp(slope, intercept, sigma, x, y):
2     lp = torch.tensor(0.0)
3
4     lp += dist.Normal(0., 3.).log_prob(slope)
5
6     lp += dist.Normal(0., 3.).log_prob(intercept)
7
8     lp += dist.HalfCauchy(1).log_prob(sigma)
9
10    for i in range(len(x)):
11        lp += dist.Normal(slope * x[i] + intercept, sigma).log_prob(y[i])
12
13    return lp
```


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

```
1 # Turing
2 @model function linreg(x, y)
3     slope ~ Normal(0,3)
4     intercept ~ Normal(0,3)
5     sigma ~ InverseGamma(1,1)
6     for i in eachindex(y)
7         y[i] ~ Normal(slope * x[i] + intercept, sigma)
8     end
9 end

1 # Pyro
2 def linreg(x, y):
3     slope = pyro.sample("slope", dist.Normal(0,3))
4     intercept = pyro.sample("intercept", dist.Normal(0,3))
5     sigma = pyro.sample("sigma", dist.HalfCauchy(1))
6
7     for i in range(len(x)):
8         pyro.sample(
9             f"y[{i}]",
10             dist.Normal(slope * x[i] + intercept, sigma),
11             obs=y[i])
```

Representation-Based

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

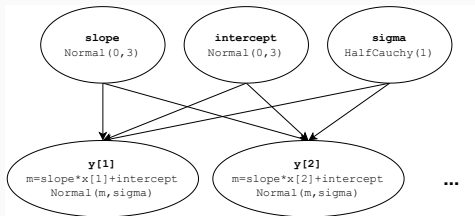
Evaluation-Based

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

Representation-Based: Graphical Models

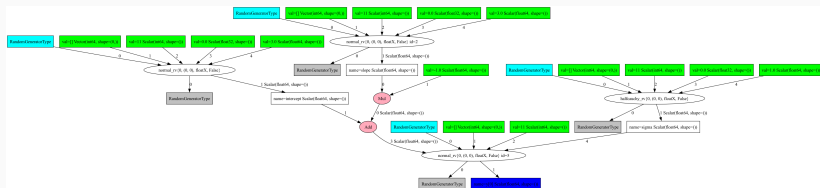
Example: PyMC

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



Representation-Based: Graphical Models

Underlying Computation Graph in PyMC



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;  
}
```

model: $\mathbb{R}^3 \rightarrow \mathbb{R}$

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;
}
```

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

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),
 ...}
```

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
```

```
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)
```

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.unwr
          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
    end
    return (var"##retval#364", __varinfo__)
  end
end
```

Strengths

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

Constraints

- General-purpose inference is difficult

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

```
1 class Distribution:
2     def sample(self, sample_shape: torch.Size = torch.Size()) -> torch.Tensor:
3         raise NotImplementedError
4
5     def log_prob(self, value: torch.Tensor) -> torch.Tensor:
6         raise NotImplementedError
```

PPL Core: Sample Context

```
1  _SAMPLE_CONTEXT = None
2
3  class SampleContext(ABC):
4      def __enter__(self):
5          global _SAMPLE_CONTEXT
6          _SAMPLE_CONTEXT = self
7
8      def __exit__(self, *args):
9          global _SAMPLE_CONTEXT
10         _SAMPLE_CONTEXT = None
11
12     @abstractmethod
13     def sample(self,
14                 address: str,
15                 distribution: dist.Distribution,
16                 observed: Optional[torch.Tensor] = None) -> torch.Tensor:
17         raise NotImplementedError
```

PPL Core: Sample Context

```
1 ctx = MySampleContext()
2 with ctx:
3     do_something()
```

```
1 ctx = MySampleContext()
2 ctx.__enter__() # global _SAMPLE_CONTEXT = ctx
3 do_something()
4 ctx.__exit__() # global _SAMPLE_CONTEXT = None
```

PPL Core: Sample Statement

```
1 def sample(address: str,  
2           distribution: dist.Distribution,  
3           observed: Optional[torch.Tensor] = None) -> torch.Tensor:  
4     global _SAMPLE_CONTEXT  
5  
6     # default behavior  
7     if _SAMPLE_CONTEXT is None:  
8         if observed is not None:  
9             return observed  
10        return distribution.sample()  
11  
12    # context specific behavior  
13    return _SAMPLE_CONTEXT.sample(address, distribution, observed)
```

Example Program

```
1 def noisy_geometric(p):
2     x = 0
3     while True:
4         b = sample(f"b_{x}", dist.Bernoulli(p))
5         if b:
6             break
7         x += 1
8     y = sample("y", dist.Normal(x,1), observed=torch.tensor(3.))
9     return x
10
```

```
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

```
1  class LogProb(SampleContext):
2      def __init__(self, trace):
3          self.log_prob = torch.tensor(0.)
4          self.trace = trace
5
6      def sample(self,
7                  address: str,
8                  distribution: dist.Distribution,
9                  observed: Optional[torch.Tensor] = None) -> torch.Tensor:
10
11         if observed is not None:
12             value = observed
13         else:
14             value = self.trace[address]
15
16         self.log_prob += distribution.log_prob(value)
17
18         return value
19
```

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

```
1  class Trace(SampleContext):
2      def __init__(self):
3          self.trace = {}
4
5      def sample(self,
6                 address: str,
7                 distribution: dist.Distribution,
8                 observed: Optional[torch.Tensor] = None) -> torch.Tensor:
9
10         if observed is not None:
11             value = observed
12         else:
13             value = distribution.sample()
14
15         self.trace[address] = {
16             'value': value,
17             'distribution': distribution,
18             'is_observed': observed is not None,
19             'log_prob': distribution.log_prob(value)
20         }
21
22         return value
23
```

Example Context (cont.)

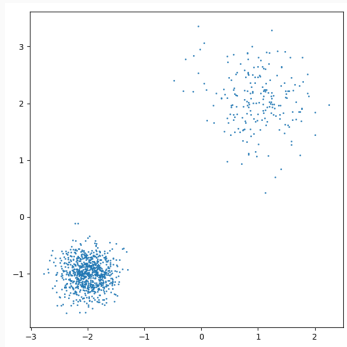
```
torch.manual_seed(0)
ctx = Trace()
with ctx:
    x = noisy_geometric(0.25)
ctx.trace
```

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, $k = 0, 1$:
 $\mu_x^k \sim \text{Uniform}(-3,3)$, $\mu_y^k \sim \text{Uniform}(-3,3)$
- Cluster spread, $k = 0, 1$:
 $\sigma^k \sim \text{InverseGamma}(1,1)$
- Cluster membership, $i = 1, \dots, N$:
 $z_i \sim \text{Bernoulli}(p)$
- Observed data, $i = 1, \dots, N$:
 $x_i \sim \text{Normal}(\mu^{z_i}, \sigma^{z_i})$

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

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:

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

$$x_i \sim \text{Bernoulli}(p)$$

```
1 def rejection_sampling(xs_observed):
2     while True:
3         # generate samples from joint
4         p = dist.Uniform(0,1).sample()
5         xs = dist.Bernoulli(p).sample(xs_observed.shape)
6         if (xs == xs_observed).all():
7             return p # accept
8         # 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.*

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 .

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

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

First General-Purpose Inference

Importance Sampling

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

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

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

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

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 $r(\Theta) = 1$

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

Thus,

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

First General-Purpose Inference

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

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

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

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

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

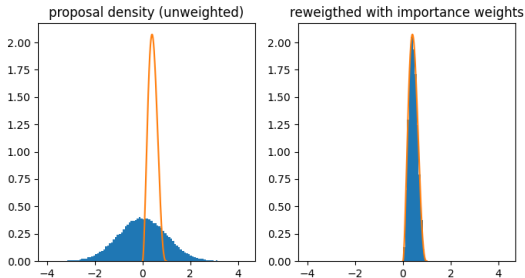
Example

Coin model

```
1 def prior(p):
2     return dist.Uniform(0,1, validate_args=False).log_prob(p).exp()
3
4 def likelihood(xs,p):
5     lp = torch.tensor(0.)
6     for x in xs:
7         lp += dist.Bernoulli(p, validate_args=False).log_prob(x)
8     return lp.exp()
9
10 def joint(xs, p):
11     return prior(p) * likelihood(xs, p)
12
13 xs = torch.tensor([0.,1.,1.,0.,0.])
14
```

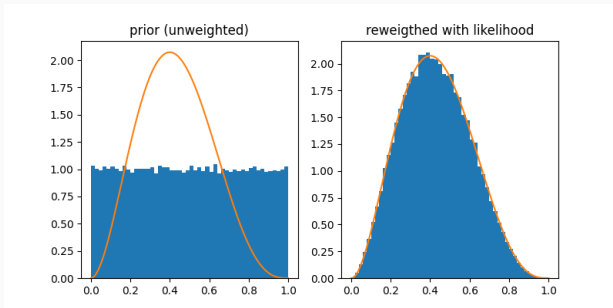
Example

```
1 proposal = dist.Normal(0,1)
2 proposal_sample = proposal.sample((N,))
3
4 weights = (
5     torch.hstack([joint(xs, proposal_sample[i]) for i in range(N)]) /
6     proposal.log_prob(proposal_sample).exp()
7 ) # importance sampling
8
9 axs[0].hist(proposal_sample, density=True, bins=100)
10 axs[1].hist(proposal_sample, density=True, bins=100, weights=weights)
```



Example

```
1 proposal = dist.Uniform(0,1) # prior
2 proposal_sample = proposal.sample((N,))
3
4 weights = torch.hstack([likelihood(xs, proposal_sample[i]) for i in range(N)])
5 # likelihood weighting
6
7 axs[0].hist(proposal_sample, density=True, bins=100)
8 axs[1].hist(proposal_sample, density=True, bins=100, weights=weights)
9
```



Example

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

```
1 def coin_model(xs):  
2     p = sample("p", dist.Uniform(0,1))  
3     for i in range(len(xs)):  
4         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*](https://www.youtube.com/watch?v=0w_4QcvBYII)

Intuition behind Bayesian inference

[*https://www.youtube.com/watch?v=yvWlpwnT1nw*](https://www.youtube.com/watch?v=yvWlpwnT1nw)

Bayesian posterior sampling

[*https://www.youtube.com/watch?v=EHqU9LE9tg8*](https://www.youtube.com/watch?v=EHqU9LE9tg8)

Generative Models

[*http://probmods.org/chapters/generative-models.html*](http://probmods.org/chapters/generative-models.html)

AI That Understands the World, Using Probabilistic Programming

[*https://www.youtube.com/watch?v=8j2S7BRRWus*](https://www.youtube.com/watch?v=8j2S7BRRWus)

A Personal Viewpoint on Probabilistic Programming

[*https://www.youtube.com/watch?v=TFXcVlKqPlM*](https://www.youtube.com/watch?v=TFXcVlKqPlM)