UPPSALA UNIVERSITET

Probabilistic Programming

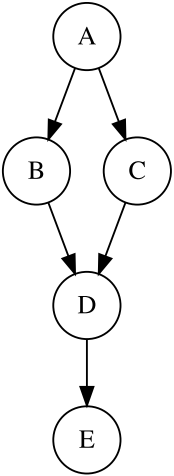
Presentation at the *Machine Learning Journal Club*

Lawrence Murray, Jan Kudlicka

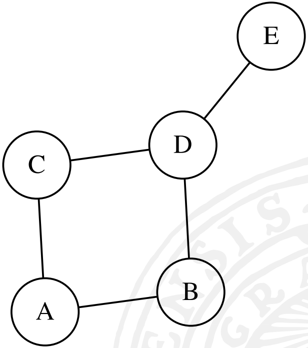
Department of Information Technology
Uppsala University

March 29, 2017

MODELS AS GRAPHS



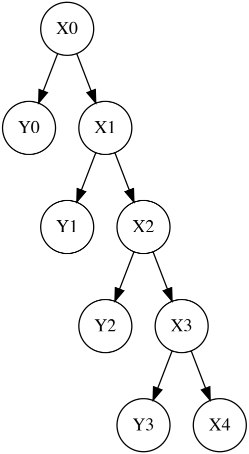
(a) Directed



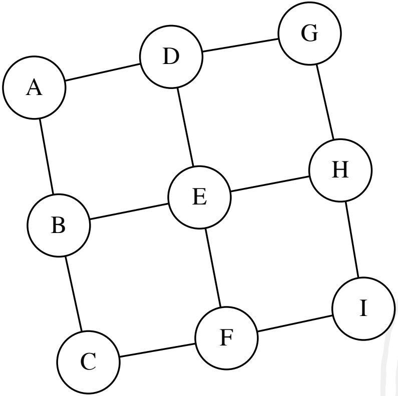
(b) Undirected

MODELS AS GRAPHS

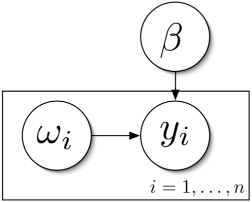
State-Space Model (SSM)



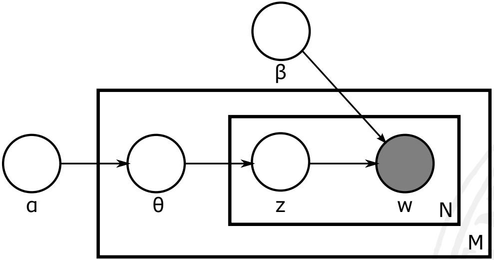
Ising Model



Bayesian Logistic Regression Model

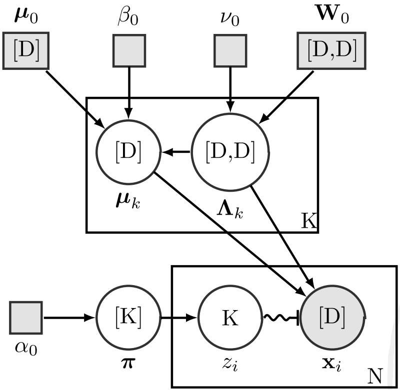


Latent Dirichlet Allocation (LDA) Model



MODELS AS GRAPHS

Gaussian Mixture Model



MODELS AS GRAPHS

- Not all models can be represented as graphical models, and the graphical language does not necessarily capture all attributes of a model.

MODELS AS GRAPHS

- Not all models can be represented as graphical models, and the graphical language does not necessarily capture all attributes of a model.

- Inference methods are often tailored for specific models, e.g. the Kalman filter for a linear-Gaussian SSM, collapsed Gibbs samplers for LDA, Polya–Gamma samplers for Bayesian logistic regression.

MODELS AS GRAPHS

- Not all models can be represented as graphical models, and the graphical language does not necessarily capture all attributes of a model.
- Inference methods are often tailored for specific models, e.g. the Kalman filter for a linear-Gaussian SSM, collapsed Gibbs samplers for LDA, Polya–Gamma samplers for Bayesian logistic regression.
- Implementations are often bespoke: of a specific inference method for a specific model.

MODELS AS PROGRAMS?

- Write a program that simulates from the joint distribution. Let this define the model.

MODELS AS PROGRAMS?

- Write a program that simulates from the joint distribution. Let this define the model.
- The program is stochastic, so that each time it runs, it may produce different output.

MODELS AS PROGRAMS?

- Write a program that simulates from the joint distribution. Let this define the model.
- The program is stochastic, so that each time it runs, it may produce different output.
- Consider constraining the output of the program, or constraining its execution.

MODELS AS PROGRAMS?

- Write a program that simulates from the joint distribution. Let this define the model.
- The program is stochastic, so that each time it runs, it may produce different output.
- Consider constraining the output of the program, or constraining its execution. **This is inference.**

MODELS AS PROGRAMS?

- Write a program that simulates from the joint distribution. Let this define the model.
- The program is stochastic, so that each time it runs, it may produce different output.
- Consider constraining the output of the program, or constraining its execution. **This is inference.**
- Programs are more expressive than graphs, because a program can do stochastic branching.

MODELS AS PROGRAMS?

- Write a program that simulates from the joint distribution. Let this define the model.
- The program is stochastic, so that each time it runs, it may produce different output.
- Consider constraining the output of the program, or constraining its execution. **This is inference.**
- Programs are more expressive than graphs, because a program can do stochastic branching. **This makes inference difficult.**

MODELS AS PROGRAMS?

- Write a program that simulates from the joint distribution. Let this define the model.
- The program is stochastic, so that each time it runs, it may produce different output.
- Consider constraining the output of the program, or constraining its execution. **This is inference.**
- Programs are more expressive than graphs, because a program can do stochastic branching. **This makes inference difficult.**
- Ideally the implementation of models is decoupled from the implementation of inference methods.

PROBABILISTIC PROGRAMMING

- *Probabilistic programming* is a programming paradigm, in the same way that object-oriented, functional and logic programming are programming paradigms.

PROBABILISTIC PROGRAMMING

- *Probabilistic programming* is a programming paradigm, in the same way that object-oriented, functional and logic programming are programming paradigms.

- *Probabilistic programming languages* (PPLs) have ergonomic support for random variables, probability distributions and inference.

PROBABILISTIC PROGRAMMING

- *Probabilistic programming* is a programming paradigm, in the same way that object-oriented, functional and logic programming are programming paradigms.
- *Probabilistic programming languages* (PPLs) have ergonomic support for random variables, probability distributions and inference.
- **The hard bit is getting a correct result.**

PROBABILISTIC PROGRAMMING

- *Probabilistic programming* is a programming paradigm, in the same way that object-oriented, functional and logic programming are programming paradigms.
- *Probabilistic programming languages* (PPLs) have ergonomic support for random variables, probability distributions and inference.
- **The hard bit is getting a correct result.**
- **The really hard bit is getting the best result.**

EXAMPLE: TWO DICE

```
die1 ~ duniform(1, 6)
die2 ~ duniform(1, 6)
sum = die1 + die2
observe sum <= 4
infer die1
```

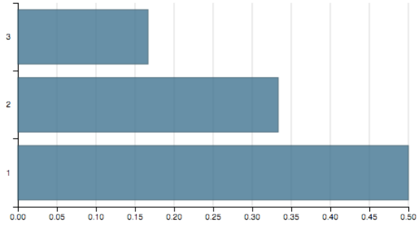
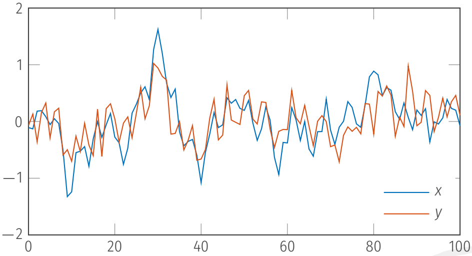


Figure generated at webppl.org.

EXAMPLE: LINEAR GAUSSIAN STATE SPACE (LGSS) MODEL

$$x_{t+1} = 0.7x_t + w$$
$$y_t = 0.5x_t + v$$
$$x_0 \sim \mathcal{N}(0, 0.1)$$
$$w \sim \mathcal{N}(0, 0.1)$$
$$v \sim \mathcal{N}(0, 0.1)$$

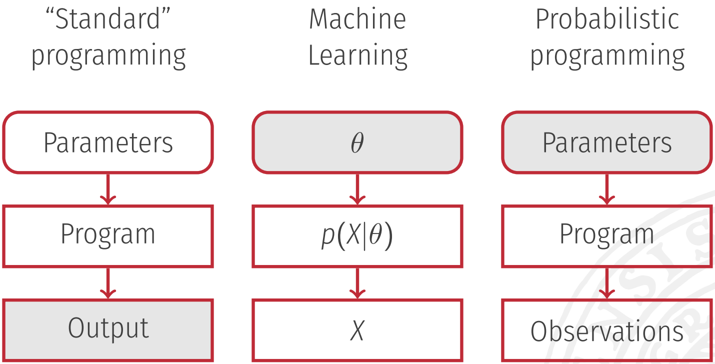



```
y = read_from_file('measurements.txt', separator='\n')
x[0] ~ normal(0, 0.1)
for t in range(100)
    observe y[t] ~ normal(0.5*x[t], 0.1)
    x[t+1] ~ normal(0.7*x[t], 0.1)
end
infer E(x[100])
```

PROBABILISTIC PROGRAMS

Probabilistic constructs in PPL:

- **Assume** – declaring and defining a random variable by specifying its probability distribution.
- **Observe** – conditioning based on a observation.
- **Infer** – calculating / estimating
  - distribution of a random variable given by an expression, or
  - its expected value, or
  - its mode(s).



Based on a figure by Frank Wood.

ADVANTAGES

- Clear separation between model and inference.
- Programs might be easier and/or quicker to “write down” than mathematical models.
- Less need for experts to find out how to do the inference and to implement it.
- Huge scope of applications (comparing to e. g. Probabilistic Graphical Models).

INFERENCE

The inference is, in general, a difficult task.

- Exact inference
  - Closed-form posterior distribution cases (e. g. Kalman filtering)
  - Enumeration – discrete models of limited dimension
- Approximate inference
  - Monte Carlo inference
  - Variational inference

USING MONTE CARLO FOR ESTIMATING EXPECTED VALUE

Monte Carlo can be used to estimate the expected value of a function of random variable:

$$I = \mathbb{E}[h(x)] = \int h(x)p(x)\mathrm{d}x.$$

Sample L points $\{x^\ell\}_{\ell=1}^L$ from $p(x)$.

$$\mathbb{E}[h(x)] \approx \hat{I}_L = \frac{1}{L}\sum_{\ell=1}^{L} h(x^\ell).$$

The law of large numbers: $\lim_{L\to\infty} \hat{I}_L = I$ with probability 1.

The central limit theorem: $\sqrt{L}(\hat{I}_L - I) \to \mathcal{N}(0, \sigma^2)$ in distribution, where $\sigma^2 = \operatorname{var} h(x)$.

IMPORTANCE SAMPLING

What if we cannot sample from $p(x)$?

Assume that

- we can evaluate
$$\tilde{p}(x) = Zp(x)$$
for all x, where Z is a (possibly unknown) constant, and
- there is another distribution $q(x)$ from which we can sample and $q(x) = 0 \Rightarrow p(x) = 0$.

We can use samples from the *proposal distribution* $q(x)$ to calculate the expected value w. r. t. $p(x)$.

IMPORTANCE SAMPLING, CONT'D

$$\mathbb{E}[h(x)] = \int h(x)p(x)\mathrm{d}x = \frac{1}{Z}\int h(x)\underbrace{\frac{\tilde{p}(x)}{q(x)}}_{w(x)}q(x)\mathrm{d}x.$$

Since $p(x)$ is a probability distribution:

$$Z = \int \tilde{p}(x)\mathrm{d}x = \int \underbrace{\frac{\tilde{p}(x)}{q(x)}}_{w(x)}q(x)\mathrm{d}x.$$

Both integrals can be estimated using Monte Carlo.

GRAPHICAL MODEL OF THE EXECUTION

Nomenclature:

- N – number of observations,
- y_n – value of the n-th observation,
- x_n – the memory state at the n-th observation,
- $g_n(y_n|x_n)$ – PDF of seeing the n-th observation y_n given the memory state x_n,
- $f_n(x_n|x_{n-1})$ – PDF of the memory state x_n given the memory state x_{n-1} at the previous observation.

GRAPHICAL MODEL OF THE EXECUTION

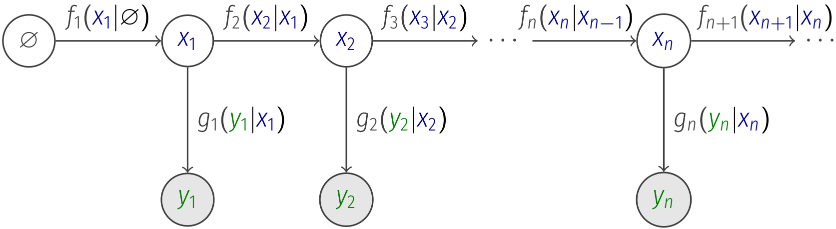
Nomenclature:

- N – number of observations,
- y_n – value of the n-th observation,
- x_n – the memory state at the n-th observation,
- $g_n(y_n|x_n)$ – PDF of seeing the n-th observation y_n given the memory state x_n,
- $f_n(x_n|x_{n-1})$ – PDF of the memory state x_n given the memory state x_{n-1} at the previous observation.

We will also use the following notation:

$$x_{1:N} = \{x_1, x_2, \ldots, x_N\}$$
$$y_{1:N} = \{y_1, y_2, \ldots, y_N\}$$

GRAPHICAL MODEL OF THE EXECUTION



$$p(x_{1:N}, y_{1:N}) = \prod_{n=1}^{N} f_n(x_n|x_{n-1}) g_n(y_n|x_n),$$

$$p(x_{1:N}|y_{1:N}) = \frac{p(x_{1:N}, y_{1:N})}{p(y_{1:N})} \propto p(x_{1:N}, y_{1:N}),$$

where $x_0 = \emptyset$.

Our interest is the posterior probability $p(x_{1:N}|y_{1:N})$.

IMPORTANCE SAMPLING REVISITED

The target distribution multiplied by an (unknown) constant:

$$\tilde{p}(x_{1:N}|y_{1:N}) = p(x_{1:N}, y_{1:N}) = \prod_{n=1}^{N} f_n(x_n|x_{n-1})g_n(y_n|x_n).$$

Let's use the following proposal distribution:

$$q(x_{1:N}) = \prod_{n=1}^{N} f_n(x_n|x_{n-1}).$$

The importance weight:

$$w = \frac{\tilde{p}}{q} = \frac{\prod_{n=1}^{N} f_n(x_n|x_{n-1})g_n(y_n|x_n)}{\prod_{n=1}^{N} f_n(x_n|x_{n+1})} = \prod_{n=1}^{N} g_n(y_n|x_n).$$

SAMPLING FROM THE PROPOSAL DISTRIBUTION

How to sample from the proposal distribution?

$$q(x_{1:N}) = \prod_{n=1}^{N} f_n(x_n | x_{n-1})$$

SAMPLING FROM THE PROPOSAL DISTRIBUTION

How to sample from the proposal distribution?

$$q(x_{1:N}) = \prod_{n=1}^{N} f_n(x_n | x_{n-1})$$

Execute the program as if it was a standard program and

- at an *assume* – sample a value from the given probability distribution
- at an *observe* – update the weight

Algorithm:

1. Sample L points $\{x^{\ell}_{1:N}\}_{\ell=1}^{L}$ from the proposal distribution $q(x)$.

IMPORTANCE SAMPLING REVISITED, CONT'D

Algorithm:

1. Sample L points $\{x^{\ell}_{1:N}\}_{\ell=1}^{L}$ from the proposal distribution $q(x)$.
2. Calculate the importance weights

$$w^{\ell} = \prod_{n=1}^{N} g_n(y_n | x_n^{\ell})$$

for $\ell = 1, \ldots, L$.

IMPORTANCE SAMPLING REVISITED, CONT'D

Algorithm:

1. Sample L points $\{x_{1:N}^{\ell}\}_{\ell=1}^{L}$ from the proposal distribution $q(x)$.
2. Calculate the importance weights

$$w^{\ell} = \prod_{n=1}^{N} g_n(y_n | x_n^{\ell})$$

   for $\ell = 1, \ldots, L$.
3. Estimate the expected value:

$$\mathbb{E}[h(x_{1:N})] \approx \frac{1}{\sum_{\ell=1}^{L} w^{\ell}} \sum_{\ell=1}^{L} w^{\ell} h(x_{1:N}^{\ell}).$$

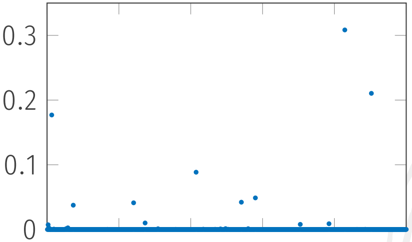
SEQUENTIAL IMPORTANCE SAMPLING (SIS)

Algorithm:

`for` $\ell = 1, \ldots, L$
  $w^\ell = 1$
  start the program
  `for` $n = 1, \ldots, N$
    continue running the program until `observe` y_n
    $w^\ell = w^\ell * g_n(y_n | x_n^\ell)$
  `end`
  continue running the program until the end
  h^ℓ = value of the inference expression
`end`
$w = w / \sum_\ell w^\ell$
$\mathbb{E}[h] = \sum_\ell w^\ell * h^\ell$

WEIGHT DEGENERACY

Showstopper:
Weight degeneracy – in real applications, almost all weights w^ℓ are zero and the value of interest must be calculated using only a few samples.



Weigths for the example from slide 12, $L = 1000$

BOOTSTRAP PARTICLE FILTER

The most basic particle filter / *Sequential Monte Carlo* (SMC) algorithm.

BOOTSTRAP PARTICLE FILTER

The most basic particle filter / *Sequential Monte Carlo* (SMC) algorithm.

Run L copies of the program (called *particles*) in parallel. At each `observe` we will resample the particles:

1. Wait until all particles have reached the `observe`.

BOOTSTRAP PARTICLE FILTER

The most basic particle filter / *Sequential Monte Carlo* (SMC) algorithm.

Run L copies of the program (called *particles*) in parallel. At each `observe` we will resample the particles:

1. Wait until all particles have reached the `observe`.
2. Calculate $w^\ell = g_n(y_n|x_n^\ell)$ for each particle.

BOOTSTRAP PARTICLE FILTER

The most basic particle filter / *Sequential Monte Carlo* (SMC) algorithm.

Run L copies of the program (called *particles*) in parallel. At each `observe` we will resample the particles:

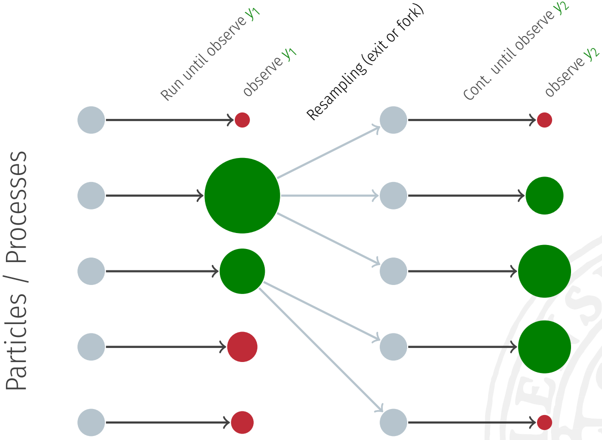
1. Wait until all particles have reached the `observe`.
2. Calculate $w^\ell = g_n(y_n|x_n^\ell)$ for each particle.
3. Sample the offspring counts $\{o^\ell\}_{\ell=1}^L$ from the multinomial distribution with the number of trials L and the event probabilities $\{w^\ell / \sum w^\ell\}_{\ell=1}^L$.
   - If $o^\ell = 0$, kill the particle process.
   - If $o^\ell = 1$, continue the process.
   - If $o^\ell > 1$, fork the process $o^\ell - 1$ times and continue.

BOOTSTRAP PARTICLE FILTER, CONT'D

Algorithm:

Start L copies of the program
for $n = 1, \dots, N$
  continue running all copies until **observe** y_n
  wait until all copies calculate $w^\ell = g_n(y_n | x_n^\ell)$
  if $n < N$
    sample $\{o^\ell\}_{l=1}^{L}$ as described above
    for $\ell = 1, \dots, L$
      if $o^\ell = 0$
        kill the process
      else if $o^\ell > 1$
        fork the process $o^\ell - 1$ times
      end
    end
  end
end
continue running all copies until the end
wait until all copies calculate $h^\ell =$ value of the inference expression
$\mathbb{E}[h] = \sum_\ell w^\ell * h^\ell / \sum_\ell w^\ell$

BOOTSTRAP PARTICLE FILTER, CONT'D



OTHER ALGORITHMS

- Metropolis-Hastings algorithm
- Hamiltonian Monte Carlo
- Gibbs sampling

Birch

Other probabilistic programming languages:

Anglican, Church, Stan, Infer.NET, WebPPL, Venture, Turing.jl, Edward

MORE INFORMATION

probabilistic-programming.org

Questions?