COMS30035, Machine learning: PGMs for Bayesian Machine Learning

James Cussens
james.cussens@bristol.ac.uk

School of Computer Science
University of Bristol

25th September 2023

The Bayesian approach

- Conceptually the Bayesian approach is easy: the goal is to compute the posterior distribution $P(\theta|D = d)$ where θ is the parameter vector and d is the observed value of the data.
- We choose a prior $P(\theta)$ and assume a particular likelihood $P(D|\theta)$ and then Bayes theorem gives us $P(\theta|D = d) \propto P(\theta)P(D = d|\theta)$.
- If we choose a *conjugate prior* for $P(\theta)$, then representing and computing $P(\theta|D = d)$ is easy.

Problems for the Bayesian approach

- "For most probabilistic models of practical interest, exact inference is intractable, and so we have to resort to some form of approximation." [Bis06, p. 523].
- We want to be able to just construct whatever joint distribution $P(\theta, D)$ we think best models the data-generating process and then compute $P(\theta|D = d)$.
- However, with this flexibility there is a price: we may not even be able to represent $P(\theta|D = d)$ easily, let alone compute it.

Problems for the Bayesian approach

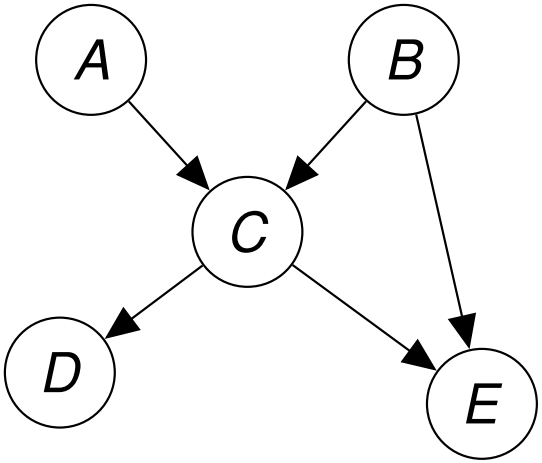
- "For most probabilistic models of practical interest, exact inference is intractable, and so we have to resort to some form of approximation." [Bis06, p. 523].
- We want to be able to just construct whatever joint distribution $P(\theta, D)$ we think best models the data-generating process and then compute $P(\theta|D = d)$.
- However, with this flexibility there is a price: we may not even be able to represent $P(\theta|D = d)$ easily, let alone compute it.
- The solution is to give up on getting $P(\theta|D = d)$ exactly and instead draw samples (of θ) from $P(\theta|D = d)$ which will allow us to approximately compute any posterior quantities, e.g. the mean of $P(\theta|D = d)$.

Univariate sampling

- We will assume throughout that we have some mechanism for sampling from any *univariate* distribution.
- There are functions for sampling from a bunch of different distributions in Python's random module. Also, to sample from a Gaussian you can use `numpy.random.normal`.
- If a multivariate distribution is described by a Bayesian network then we can use *ancestral sampling* to sample a joint instantiation of the variables.

Ancestral sampling

$$p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|C)p(E|B, C)$$



- Just ensure that we sample values for all parents of a node before we sample a value for that node (this is always possible due to acyclicity).
- So to sample from $p(A, B, C, D, E)$ we first sample values for A and B, suppose we get the values $A = 0, B = 1$. We then sample a value for C from the conditional distribution $P(C|A = 0, B = 1)$, and so on. [Bis06, §8.1.2].

Sampling from marginal and conditional distributions

$$p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|C)p(E|B, C)$$

- We can approximate any marginal distribution (say, $P(B, E)$) by sampling full joint instantiations (by e.g. ancestral sampling) and then only keeping the values of the variables in the marginal.
- We can use *rejection sampling* to sample from conditional distributions.
- For example, to sample from $P(B, D|E = 1)$ we sample from the marginal distribution $P(B, D, E)$ and throw away those samples where $E \neq 1$.
- Rejection sampling is typically inefficient.

Approximating expectations

- Often we want to compute expected values with respect to some posterior distribution [Bis06, p. 524].

$$E[f] = \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} \tag{1}$$

- If we draw independent samples $\mathbf{z}^{(l)}$, $l = 1, \ldots, L$ from $p(\mathbf{z})$ then we can approximate $E[f]$ as follows:

$$\hat{f} = \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^{(l)}) \tag{2}$$

Markov chain Monte Carlo

- If we can sample from a distribution then we have a simple way to compute approximate values. But what if we cannot?

Markov chain Monte Carlo

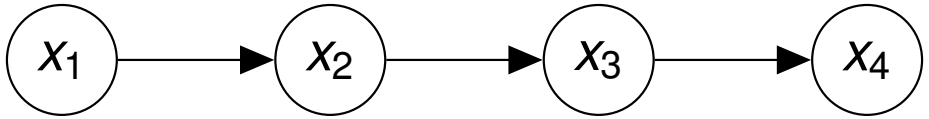
- If we can sample from a distribution then we have a simple way to compute approximate values. But what if we cannot?
- If we can sample from *a sequence of distributions* which eventually reaches (or gets very close to) the desired distribution, then we can adopt the following strategy:
  1. Draw a sample from each distribution in this sequence.
  2. Only keep the samples once we get 'close enough' to the desired distribution.
- This is the approach of Markov chain Monte Carlo (MCMC).

Markov chains

"A first-order Markov chain is defined to be a series of random variables $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(M)}$ such that the following conditional independence property holds for $m \in \{1, \ldots, M-1\}$" [Bis06, p. 539].

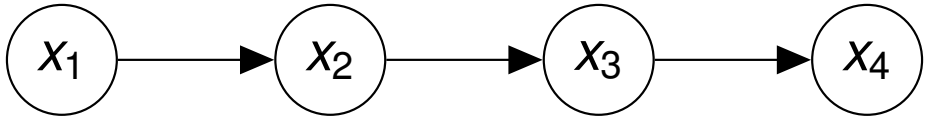
$$p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}) = p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)}) \tag{3}$$

- $\mathbf{z}^{(m)}$ often represents (or can be imagined to represent) the mth state of some dynamic system so that $p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)})$ is a *state transition probability*.
- If $p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)})$ is the same for all m then the chain is *homogeneous*.
- (We also need an *initial distribution* $p(\mathbf{z}^{(1)})$.)
- Here's the Bayesian network representation of a Markov chain where $M = 4$.



- Sampling from a Markov chain is easy: it's just a special case of ancestral sampling.

Markov chain Monte Carlo



- A Markov chain defines a sequence of marginal distributions; for the BN above these are $P(x_1)$, $P(x_2)$, $P(x_3)$ and $P(x_4)$.
- The goal of MCMC is to design a Markov chain so that this sequence of marginal distributions converges on the distribution we want.
- Then we can just sample from the Markov chain and only keep the sampled values of the 'later' random variables.
- The sampled values we draw are **not** independent, but this is a price we have to pay.

How to get MCMC to work?

- We have a clear goal: **given** a target probability distribution $p(\mathbf{z})$, **construct** a Markov chain $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(i)} \ldots$ such that $\lim_{i \to \infty} p(\mathbf{z}^{(i)}) = p(\mathbf{z})$.
- (For Bayesian machine learning the target distribution will be $P(\theta | D = d)$, the posterior distribution of the model parameters given the observed data.)
- One solution to this is the *Metropolis-Hastings* algorithm.

The Metropolis-Hastings (MH) algorithm

- We define a single transition probability distribution for a homogeneous Markov chain.
- Let the current state be $\mathbf{z}^{(\tau)}$. When using the MH algorithm sampling the next state happens in two stages:
  1. We generate a value $\mathbf{z}^*$ by sampling from a *proposal distribution* $q(\mathbf{z}|\mathbf{z}^{(\tau)})$.
  2. We then accept $\mathbf{z}^*$ as the new state with a certain *acceptance probability*. If we don't accept $\mathbf{z}^*$ then we 'stay where we are', so that $\mathbf{z}^{(\tau)}$ is both the old and new state.

The Metropolis-Hastings acceptance probability

Let $p(\mathbf{z})$ be the *target distribution*. The acceptance probability is: [Bis06, p. 541].

$$A(\mathbf{z}^*, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{p(\mathbf{z}^*)q(\mathbf{z}^{(\tau)}|\mathbf{z}^*)}{p(\mathbf{z}^{(\tau)})q(\mathbf{z}^*|\mathbf{z}^{(\tau)})}\right) \tag{4}$$

- If $p(\mathbf{z}) = \tilde{p}(\mathbf{z})/Z$ then we have $p(\mathbf{z}^*)/p(\mathbf{z}^{(\tau)}) = \tilde{p}(\mathbf{z}^*)/\tilde{p}(\mathbf{z}^{(\tau)})$, so we only need p up to normalisation. This is a big win!
- If the proposal distribution is symmetric then the 'q' terms cancel out: a special case known as the *Metropolis algorithm*.
- Note that for the Metropolis algorithm if $p(\mathbf{z}^*) \geq p(\mathbf{z}^{(\tau)})$ then we always accept and 'move' to $\mathbf{z}^*$.

Does Metropolis-Hastings (always) work?

.

- It can be shown [Bis06, p. 541] that the target distribution is an *invariant distribution of the Markov chain*: if the sequence of distributions $p(\mathbf{z}^{(i)})$ reaches the target distribution then it stays there.
- Also, typically the Markov chain does converge to the target distribution.
- The *rate* at which we converge to the target distribution is greatly influenced by the choice of proposal distribution.

MCMC in practice

- Straightforward Metropolis-Hastings is not the state-of-the-art in MCMC.
- *Probabilistic programming* systems like PyMC by default use more sophisticated MCMC algorithms (to avoid getting stuck).
- From the PyMC intro overview: "Probabilistic programming (PP) allows flexible specification of Bayesian statistical models in code. PyMC is a PP framework with an intuitive and readable, yet powerful, syntax that is close to the natural syntax statisticians use to describe models. It features next-generation Markov chain Monte Carlo (MCMC) sampling algorithms such as the No-U-Turn Sampler"
- When using MCMC we (1) throw away early samples ('burn-in') and (2) 'run independent chains' to check for convergence.
- PyMC uses $\hat{R}$ (`r_hat`) to check for convergence; this value should be close to 1.

Let's do some Bayesian machine learning with PyMC!

- I've found the easiest way to get the introductory Jupyter notebooks mentioned in the PyMC website is to clone the PyMC github repo.
- You can then find them in `pymc/docs/source/learn/core_notebooks`

Christopher M. Bishop.
Pattern Recognition and Machine Learning.
Springer, 2006.