
CSE 291d. Assignment 3

Out: Thu Jan 25

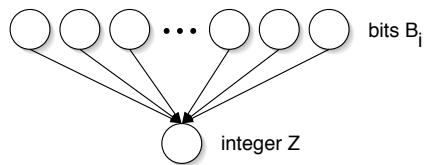
Due: Fri Feb 2

3.1 Stochastic simulation

Consider the belief network shown below, with n binary random variables $B_i \in \{0, 1\}$ and an *integer* random variable Z . Let $f(B) = \sum_{i=1}^n 2^{i-1} B_i$ denote the nonnegative integer whose binary representation is given by $B_n B_{n-1} \dots B_2 B_1$. Suppose that each bit has prior probability $P(B_i = 1) = \frac{1}{2}$, and that

$$P(Z|B_1, B_2, \dots, B_n) = \left(\frac{1 - \alpha}{1 + \alpha} \right)^{\alpha^{|Z - f(B)|}}$$

where $0 < \alpha < 1$ is a parameter measuring the amount of noise in the conversion from binary to decimal. (Larger values of α indicate greater levels of noise.)



- Show that the conditional distribution for binary to decimal conversion is normalized; namely, that $\sum_z P(Z = z | B_1, B_2, \dots, B_n) = 1$, where the sum is over all integers $z \in [-\infty, +\infty]$.
- Use the method of *likelihood weighting* to estimate the probability $P(B_6 = 1 | Z = 32)$ for a network with $n = 10$ bits and noise level $\alpha = 0.15$. Turn in your source code, as well as a plot of your estimate as a function of the number of samples.

You may program in the language of your choice, and you may use any program at your disposal to plot the results. (MATLAB works extremely well for both.)

3.2 Markov model

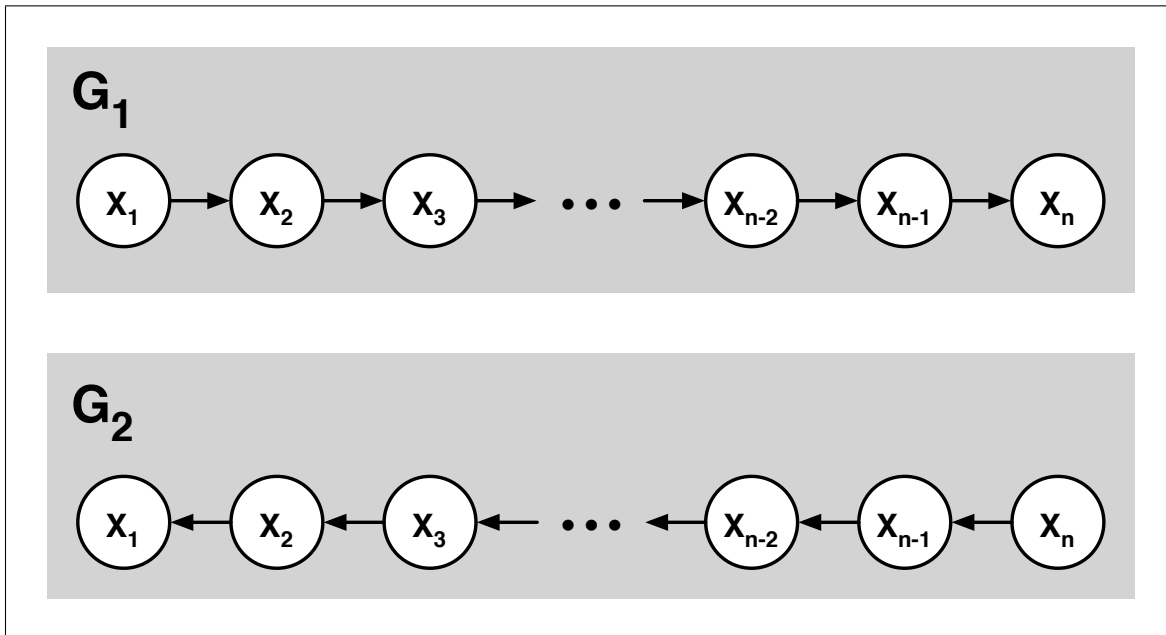
Consider the first-order Markov model with random variables X_t and $n \times n$ transition matrix:

$$U_{ij} = P(X_{t+1} = j | X_t = i).$$

- Show that $P(X_t = j | X_0 = i) = [U^t]_{ij}$, where U^t is the t^{th} power of the transition matrix.
 - Consider the computational complexity of this inference. Devise a simple algorithm, based on matrix-vector multiplication, that scales as $O(n^2 t)$.
 - Show alternatively that the inference can also be done in $O(n^3 \log_2 t)$.
-

3.3 Maximum likelihood estimation

Consider the two DAGs shown below, G_1 and G_2 , over the same nodes $X = (X_1, X_2, \dots, X_n)$, that differ only in the direction of their edges.



Suppose that the CPTs for these DAGs are obtained by maximum likelihood estimation from a “fully observed” data set in which each example provides a complete instantiation of the nodes in the DAG. In this data set, let $N_i(x)$ count the number of examples in which $X_i = x$, and let $N_i(x, x')$ count the number of examples in which $X_i = x$ and $X_{i+1} = x'$.

- Express the maximum likelihood estimates for the CPTs in G_1 in terms of these counts.
 - Express the maximum likelihood estimates for the CPTs in G_2 in terms of these counts.
 - Using your answers from parts (a) and (b), show that the maximum likelihood CPTs for G_1 and G_2 from this data set give rise to the same joint distribution over X .
-

3.4 Convergence of gradient descent

Minimizing a function $f(\vec{x})$ by gradient descent is based on the simple iterative update:

$$\vec{x}_{n+1} = \vec{x}_n - \eta \nabla f,$$

where $\eta > 0$ is a small positive learning rate, \vec{x}_n is the search location at the n th iteration, and ∇f is the gradient of $f(\vec{x})$ evaluated at \vec{x}_n . One way to gain intuition for gradient descent is to analyze its behavior in simple settings. Perhaps the simplest setting is minimization of a one-dimensional function $f(x)$ over the real line $x \in \mathcal{R}^1$. The learning rule for gradient descent in this setting is simply:

$$x_{n+1} = x_n - \eta f'(x_n).$$

- (a) Consider minimizing the function $f(x) = \frac{\alpha}{2}(x - x_*)^2$ by gradient descent, where $\alpha > 0$. Derive an expression for the error $\varepsilon_n = x_n - x_*$ at the n^{th} iteration in terms of the initial error ε_0 and the step size $\eta > 0$.
- (b) For what values of the step size η does the update rule converge to the minimum at x_* ? What step size leads to the fastest convergence?

In practice, the gradient descent learning rule is often modified to dampen oscillations at the end of the learning procedure. A common variant of gradient descent involves adding a so-called *momentum* term:

$$\vec{x}_{n+1} = \vec{x}_n - \eta \nabla f + \beta (\vec{x}_n - \vec{x}_{n-1}),$$

where $\beta > 0$. Intuitively, the name arises because the optimization continues of its own momentum (stepping in the same direction as its previous update) even when the gradient vanishes. In one dimension, this learning rule simplifies to:

$$x_{n+1} = x_n - \eta f'(x_n) + \beta(x_n - x_{n-1}).$$

- (c) Consider minimizing the quadratic function in part (a) by gradient descent with a momentum term. Again, let $\varepsilon_n = x_n - x_*$ denote the error at the n th iteration. Show that the error in this case satisfies the recursion relation:

$$\varepsilon_{n+1} = (1 - \alpha\eta + \beta)\varepsilon_n - \beta\varepsilon_{n-1}.$$

- (d) Suppose that the second derivative $\alpha = f''(x^*)$ is given by $\alpha = 1$, the learning rate by $\eta = \frac{4}{9}$, and the momentum parameter by $\beta = \frac{1}{9}$. Show that one solution to the recursion in part (c) is given by:

$$\varepsilon_n = c^n \varepsilon_0,$$

where ε_0 is the initial error and c is a numerical constant to be determined. (Other solutions are also possible, depending on the way that the momentum term is defined at time $t = 0$; do not concern yourself with this.) How does this rate of convergence compare to that of gradient descent with the same learning rate ($\eta = \frac{4}{9}$) but no momentum parameter ($\beta = 0$)?

3.5 Newton's method

Newton's method is another iterative optimization procedure. Consider a twice-differentiable function $f(x)$ over the real line. Newton's method for computing a local minimum of this function is:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

Note that this update rule uses information from first and second derivatives (which often makes it much faster than gradient descent). Also, it does not involve a learning rate.

- (a) Consider the function $f(x) = x_* \log(x_*/x) - x_* + x$, where $x_* > 0$. Show that the minimum occurs at $x = x_*$, and sketch the function in the region $|x - x_*| < x_*$.
 - (b) Consider minimizing the function in part (a) by Newton's method. Derive an expression for the relative error $r_n = (x_n - x_*)/x_*$ at the n^{th} iteration in terms of the initial relative error r_0 . Note the rapid convergence (which is typical of Newton's method). What is the radius of convergence?
-