EE3731C Tutorial - Statistical Signal 3.5

Department of Electrical and Computer Engineering

1. The point of this problem is to demonstrate that the computation of MAP and MMSE estimates are much more difficult when x is multi-dimensional. This combinatorial explosion is known as the curse of dimensionality.

   (a) Since x_n can take on K values and $n = 1, \cdots, N$, there are K^N possible values of $\vec{x}$. This means that we need to compare K^N possible values of $p(\vec{x}|y)$ in order to find the best $\vec{x}$. Since $N = 20$, $K = 10$ and the evaluation of $p(\vec{x}|y)$ takes 1ms, the exhaustive search for the MAP estimate will take at least $10^{20} \times 1\text{ms} = 10^{17}\text{s} \approx 3$ billion years. For comparison, the age of our planet Earth is roughly 4.5 billion years!

   (b) On the surface, $x_n^{MMSE} = E_{p(x_n|y)}(x_n) = \sum_{x_n=1}^{K} x_n p(x_n|y)$ seems easy to compute because we are just adding and multiplying K numbers. The difficult part comes from computing $p(x_n|y)$ from $p(\vec{x}|y)$. For example, to compute $p(x_1|y)$, we use the law of total probability:

   $$p(x_1|y) = \sum_{x_2=1}^{K}\sum_{x_3=1}^{K}\cdots\sum_{x_N=1}^{K} p(x_1, \cdots, x_N|y) \tag{1}$$

   Therefore to compute $p(x_1|y)$ in Eq. (1), we have to add K^{N-1} different values of $p(\vec{x}|y)$. Assuming addition and multiplication are much faster than evaluating $p(\vec{x}|y)$, then Eq. (1) would take $K^{N-1} \times 1\text{ms}$. Since we have to do this for K different values of x_1, the final computation cost for computing x_1^{MMSE} is K^Nms. Since $N = 20$, $K = 10$, computing x_1^{MMSE} requires roughly 3 billion years. To compute $\vec{x}^{MMSE}$ would require roughly $3N = 60$ billion years.

Therefore we see that even for relatively small problems, the computational cost explodes very quickly as the dimensionality increases. In our programming assignment, $\vec{x}$ is the mapping between the original alphabets and the encrypted alphabets. In that case, $N = 27$ and $K = 27$, which are bigger than the example we considered here. In real world problems, N and K might be millions or billions.

In contrast, suppose we use the Metropolis algorithm to generate M samples from $p(\vec{x}|y)$. Let's denote these samples as $\vec{x}^{(1)}, \cdots, \vec{x}^{(M)}$. We can then approximate MAP estimation by

$$\vec{x}_{MAP} \approx \operatorname*{argmax}_{\vec{x}^{(m)}} p(\vec{x}^{(m)}|y),$$

which only requires us to compare M instances of $p(\vec{x}|y)$. We can also approximate MMSE estimation by

$$\vec{x}^{MMSE} \approx \frac{1}{M} \sum_{m=1}^{M} \vec{x}^{(m)},$$

which only requires us to average M numbers.

In the biased dice example (from lecture), we run the Metropolis algorithm for 10 iterations in order to generate 1 sample. We then repeat this 1000 times to generate 1000 samples. In the programming assignment, we will run the Metropolis algorithm for 15000 iterations in order to generate 1 sample. As will be seen in the assignment, this 1 sample is good enough to unscramble the encrypted message. We actually do not need to generate multiple samples in order to approximate the MAP estimate (even though we could do so).

2. First observe that $q(x'|x) = q(x|x')$, so this is a valid proposal distribution.

   (a) Since $p(x = 3) > p(x = 2)$, we accept x' with probability 1.

   (b) Since $p(x = 2) < p(x = 3)$, we accept x' with probability $p(x = 2)/p(x = 3) = 1/2$.

   (c)

$$T = \begin{bmatrix} 11/12 & 1/12 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 0 & 1/6 & 1/2 & 1/3 \\ 0 & 0 & 2/9 & 7/9 \end{bmatrix}$$

   To see this

   -
$$\begin{aligned} T(1,2) &= q(x' = 2|x = 1)p(x' \text{ accepted}|x' = 2 \text{ and } x = 1) \\ &= \frac{1}{3} \times \frac{1}{4} = 1/12 \\ T(1,1) &= 1 - T(1,2) - T(1,3) - T(1,4) \\ &= 1 - 1/12 - 0 - 0 = 11/12 \end{aligned}$$

   - Since $p(x = 2)$ is smaller than $p(x = 1)$ and $p(x = 3)$, therefore $T(2, x') = q(x'|x)$.
   -
$$\begin{aligned} T(3,2) &= q(x' = 2|x = 3)p(x' \text{ accepted}|x' = 2 \text{ and } x = 3) \\ &= 1/3 \times 1/2 = 1/6 \\ T(3,4) &= q(x' = 4|x = 3)p(x' \text{ accepted}|x' = 4 \text{ and } x = 3) \\ &= 1/3 \times 1 = 1/3 \\ T(3,3) &= 1 - T(3,1) - T(3,2) - T(3,4) = 1/2 \end{aligned}$$

- $$\begin{aligned} T(4,3) &= q(x'=3|x=4)p(x' \text{ accepted}|x'=3 \text{ and } x=4) \\ &= \frac{1}{3} \times \frac{2}{3} = 2/9 \\ T(4,4) &= 1 - T(4,1) - T(4,2) - T(4,3) \\ &= 1 - 0 - 0 - 2/9 = 7/9 \end{aligned}$$

(d) In the equivalent Markov chain (see T from previous part), for $n \geq 3$ (i.e., after 3 time steps), probability of going from any initial x to any other x is non-zero. Therefore the equivalent Markov chain satisfies the conditions of the Fundamental Theorem of Markov Chain and the Markov chain will converge to $p(x)$.

As a check, we can take the matrix T and find the left eigenvector of T with eigenvalue equal to 1. In matlab, we can do this, by using the "eig" function:
» T = [11/12 1/12 0 0; 1/3 1/3 1/3 0; 0 1/6 1/2 1/3; 0 0 2/9 7/9];
» [V, D] = eig(T'); % left eigenvectors of T = right eigenvectors of T'
» V(:, 3)'/sum(V(:, 3)) % third eigenvector returned by eig has eigenvalue 1
0.4000 0.1000 0.2000 0.3000