CS664 Lecture #5 **Global Optimization** 9/18/01
Scribe: Bo Pang Lecturer: Ramin Zabih

0 Quiz #1

1 Introduction

Last time we started the discussion on image restoration via global optimization. So far we've been concentrating on local methods which give us bad answers quickly, in the following lectures, we'll examine global methods that give us good answers, though slowly.

We are facing an extremely large searching space here, consider the task of labeling pixels, suppose we have k possible labels, when label corresponds to intensity, k can have 256 possible values, to label n pixels $(p_1, p_2, \cdots, p_n)$ – n can easily be as large as 100,000 for an image – there will be $k^n(256^{100,000})$ possible labelings.

In the last lecture, we noted that there are two main problems for global maximization:

- How to choose an appropriate energy function
- How to minimize the energy function efficiently

In this lecture, we'll consider these two problems respectively.

2 Energy Function

An appropriate energy function should capture the following 2 constraints:

- Smooth
- Like/respect input data

This can be done by representing each constraint with one term and then sum them up:

$$E(f) = \underbrace{\sum_p c(p, f(p))}_{data\ term} + \lambda \underbrace{\sum_{p,q \in N} d(f(p), f(q))}_{smoothness\ term}$$

- $c(p, f(p))$ is the old familiar cost function which captures how close are the input and output
- $d(f(p), f(q))$ is the cost to have p, q as neighbors (smoothness cost), it captures the intuition that the more differences neighboring pixels have, the less smooth the output image is.

- λ is the regularization parameter, it captures the relative importance of the two terms, thus controls the trade-off between the two constraints, in computer vision, it also has this fancy name: Tikhanov regularization

Example: Set $\lambda = 1, c = d = L_2$,
consider input = [100 100 0], we have:

	Output	Data term	Smooth term	Energy
1	[100 100 0]	0	100^2	10000
2	[100 50 0]	50^2	$2 * 50^2$	7500

In this example, *output 2* has a lower energy, though we consider *output 1* as the better answer of the two. So in this case, $\lambda = 1$ might be too big, as the output that gets "over-smoothed" is preferred.

3 Energy Minimization

3.1 Energy Landscape

Consider two different energy landscapes, one is smooth as in Figure 1:

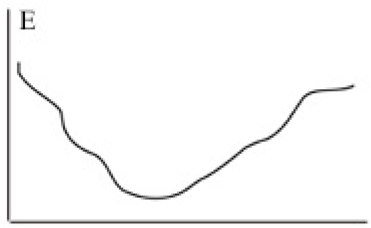

Figure 1: A Smooth Landscape

Because the energy function lacks local minimum, it's relatively easy to find the global minimum, we can just apply gradient descent or hill climbing. (Though this kind of minimization is considered to be relatively easy compared to most energy functions we come across in computer vision, it is by no means a trivial problem in itself, there's a rich literature on the minimization of convex functions.)

But if we have an energy function as shown in Figure 2, here the energy landscape is full of local minimums, by simply applying gradient descent, we can be trapped in a local minimum without reaching the global one. One method that deals with this is Simulated Annealing.

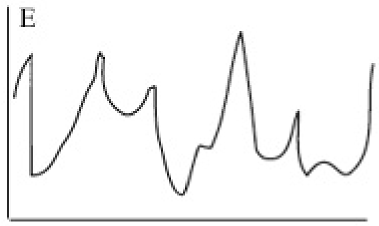

Figure 2: A Bumpy Landscape

3.2 Simulated Annealing

Claim: Simulated Annealing can find global minimum of an *arbitrary* energy function. (From this claim (*arbitrary!*), we would expect this algorithm to run very slowly (exponential time)).

We'll start by introducing Metropolis Algorithm which is the basis of Simulated Annealing.

3.3 Metropolis Algorithm

This dynamic method of generating an arbitrary probability distribution was invented by Metropolis, Tellers, and Rosenbluths in 1953 (supposedly at a Los Alamos dinner party). [1]

Algorithm: Start at somewhere f

Step 1. **[sampling]** Generate a random change to get f' (in Vision, this corresponds to changing the labeling of one pixel).

Step 2. compute $\Delta E = E(f') - E(f)$

- If $\Delta E < 0$, then this is a downhill move, accept the change (set $f = f'$ go back to Step 1.)
- If $\Delta E > 0$, accept the change with probability $e^{-\Delta E/T}$

T is the parameter to Metropolis:

The probability distribution $p(x) \propto e^{-\Delta E/T}$ is very important in statistical physics, it is called Gibbs or Boltzman distribution. Note the difference in the shape of this function as T changes (Figure 3)

[1] http://www.npac.syr.edu/users/paulc/lectures/montecarlo/node24.html

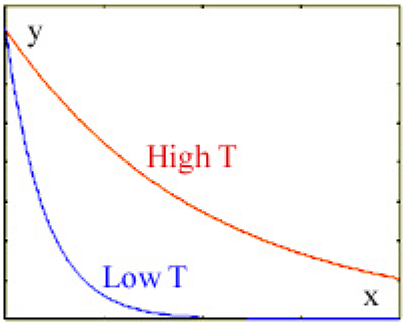

Figure 3: Gibbs Distribution with High/Low Temperature

So at one extreme(when temperature is very high), the probability to go uphill goes to constant, and we have a random search, at the other extreme(when temperature is very low), the probability to go uphill goes to zero, and we have gradient descent.

Metropolis algorithm is randomized, the output is a distribution, and this distribution is actually again the Gibbs distribution with parameter T.

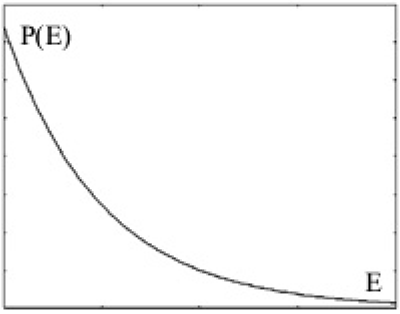

Figure 4: Distribution of the Output of Metropolis

Note: this indicates that we can have better results at low temperatures, but computation can get really slow when T is low. This result is not trivial to prove, and in order to prove that, we need to take a look on Markov Chains first.

3.4 Markov Chains

Goal: to prove the distribution of the output of Metropolis is Gibbs distribution.

Markov Chains is actually a stochastic automata, a finite state machine with a probability associated with each edge, here is an example of a Markov Chain with two states:

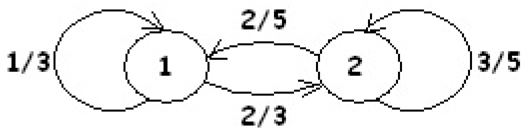

Figure 5: A Markov Chain

Note: Markov Chains has memory-less property (or Markov Property), there's no notion of history (i.e. which transition out of a certain state to take doesn't depend on how we get into it)

Consider a sequence of random variables indexed by time: $X_1, X_2, \cdots, X_n$, suppose we're throwing a dice, and the outcome can be one of a, b, c, d, e, f. This is a random process.

Markov property says:

$$P(X_i = x_i | X_{i-1} = x_{i-1}, \cdots, X_1 = x_1) = P(X_i = x_i | X_{i-1} = x_{i-1})$$

meaning the value of X_i only depends on X_{i-1}. If our random process has Markov property, we would be able decompose the joint distribution $P(X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n)$ into the product of the probability of the initial state and successive transition probabilities.

Back to our dice example, the probability of getting the outcome sequence a, c, e can be decomposed as follows:

$$P(X_1 = a, X_2 = c, X_3 = e) = P(X_1 = a)P(X_2 = c|X_1 = a)P(X_3 = e|X_2 = c)$$

Notice that the Markov Chain is described by the transition probabilities, and we can capture this by defining the **transition matrix** K:

$$k_{ij} = P(X_{m+1} = j | X_m = i)$$

Example: the Markov Chain as pictured in Fig.5 can be described by the transition matrix $K = \begin{bmatrix} 1/3 & 2/3 \\ 2/5 & 3/5 \end{bmatrix}$

Now imagine we are running a Markov Chain, suppose we start in state 1 with probability α (this implies that we start in state 2 with probability $1-\alpha$), more formally, we have:

$$P(X_1 = 1) = \alpha, \;\; P(X_1 = 2) = 1 - \alpha$$

Consider the probability of being in each state at the next step:

$$\begin{aligned} P(X_2 = 1) &= P(X_1 = 1) * P(X_2 = 1|X_1 = 1) + P(X_1 = 2) * P(X_2 = 1|X_1 = 2) \\ &= \alpha * 1/3 + (1-\alpha) * 2/5 \\ P(X_2 = 2) &= \alpha * 2/3 + (1-\alpha) * 3/5 \end{aligned}$$

Or we can express this in a more compact way using transition matrix:

$$[\, P(X_2 = 1) \quad P(X_2 = 2)\,] = [\, \alpha \quad 1-\alpha\,] \cdot K$$

And after n moves, we'll reach $[\, \alpha \quad 1-\alpha\,] \cdot K^n$
Note that as K is describing the transition probabilities, we have:

- all of its rows sum up to 1
- $\forall i,j \;\; 0 \le k_{ij} \le 1$

3.5 Ergodicity Theorem

Definition A Markov Chain is ergodic if it's

- strongly connected (can go from anywhere to anywhere else)
- every state has self-loop (any state can go back to itself)

Ergodicity Theorem If a Markov Chain is ergodic, and its transition matrix is K, then $\exists \Pi^* \; s.t. \; \Pi^* \cdot K = \Pi^*$, $\forall \Pi \; \lim_{n\to\infty} \Pi \cdot K^n = \Pi^*$

In plain English, this theorem says, if we run a Markov Chain long enough, the resulting distribution will be independent of the initial distribution, but will converge to Π^*, we call this Π^* fixed point, or attractor, stationary distribution.

To be continued... Running Matropolis algorithm is a Markov Chain, where every possible labeling is a node, and it is ergodic, since there is always a path from any labeling to another, and when a certain change is not accepted, you stay where you are.

In the next lecture, we are going to prove that the Gibbs distribution is Π^*, i.e. $\Pi_G = K \cdot \Pi_G$