Markov chains

1 Why Markov Models

We discuss Markov models now. These have relevance in a few ways:

1. Markov models are a good way to model local, overlapping sets of information, which reflect cues to our understanding of regions.
2. Markov chains provide a stochastic model of diffusion that applies to individual particles, which gives us the foundation for diffusion we've studied.
3. This stochastic diffusion also provides a useful model of the spread of information throughout an image.

2 Markov Chains

We will start by discussing the most simple Markov model, a Markov chain. We have already talked about these a little, since diffusion of a single particle can be thought of as a Markov chain. We can also use Markov chains to model contours, and they are used, explicitly or implicitly, in many contour-based segmentation algorithms. One of the key advantages of 1D Markov models is that they lend themselves to dynamic programming solutions.

In a Markov chain, we have a sequence of random variables, which we can think of as describing the state of a system, x_1, x_2, x_n. What makes them Markov is a conditional independence property that says that each state only depends on the previous state. That is:

$$P(x_n|x_1, x_2, x_{n-1}) = P(x_n|x_{n-1})$$

Diffusion of a single particle offers us a simple example of this. Let x_i be the position of a particle at time i. Suppose at each time step, the particle jumps either one unit to the left or right, or stays in the same place. We can see that x_i depends on x_{i-1}, but that if we know x_{i-1} the previous values of x are irrelevant. We can also see that this is a stochastic version of the deterministic diffusion we've studied. If there are many particles, we can use the law of large numbers to assume that a fixed fraction of them jump to the left and right. Remember that we saw that the position x_i will have a Gaussian distribution for large i, because of the central limit theorem.

Many physical processes can be modeled as Markov chains, because this just means that the new state only depends on the previous state. We can also use diffusion as a model of contour shape. This can be done according to the following equations:

$$\dot{x} = \cos\theta \quad (1)$$

$$\dot{y} = \sin\theta \quad (2)$$

$$\dot{\theta} = \hat{\kappa}(0, \sigma^2; t) \quad (3)$$

with the additional constraint that a diffusing particle disappears with a constant probability during a fixed time period.

We can view this as a prior on the shape of contours, so that if we want to reconstruct an image in which contours have been fragmented, we can do this by combining image information with a prior. First let's look at Equations 1-3. These say that a particle has a direction, θ, in which it moves, and this direction diffuses according to a Gaussian distribution. Contours are modeled as paths traced by particles undergoing this stochastic motion. σ is a parameter that controls how wiggly the curve is; with $\sigma = 0$ contours are straight lines, with large σ contours are more wiggly. Given this model, the negative log probability of a contour is:

$$E = k \int \kappa^2(t) dt$$

where k is a constant depending on σ, and the rest is the integral of the squared curvature. Something like this is often used as a model of Gestalt Good Continuation. We prefer image interpretations which minimize this energy, with the smoothest curves, which are the most likely image interpretations. If we connect line segments with a curve that is tangent to the lines at their endpoints, and that minimizes this energy, we call this the *curve of least energy*.

Note that if we have this notion of decay the probability of a contour decreases exponentially with its length, so the negative log probability becomes:

$$E = k \int \kappa^2(t) + \lambda dt \tag{4}$$

where λ is another constant that indicates the likelihood of decay. This energy also favors shorter curves; length is traded off against smoothness with the parameter λ.

Markov chains have some nice properties. One is that their steady state can be found by solving an eigenvalue problem, another is that they lend themselves to dynamic programming solutions. Let's look at these general properties.

If a Markov chain involves transitions between a discrete set of states, it's very useful to describe these transitions from state to state using vectors and matrices. Let p_j^t be the probability of being in state j and time t. Now let's put these in a vector, so that: $p^t = (p_1^t, , p_n^t)$ gives the probability distribution over all states we can be in. These are probabilities because even if the state at time 0 is deterministic, after that it will be stochastic. Notice that $\sum_j p_j^t = 1$. Now, the Markov model is completely specified by the probability that if I'm in state s_j at time $t-1$ that I'll be in state s_i at time t, for all i and j. (In principle, these probabilities might also depend on t, that is, vary over time, but I'll assume for now that they don't.) We build a matrix with these probabilities, called A, with entries A_{ij}. Notice that every column of A has to sum to 1 because if I am in state j at time $t-1$ I have to move to exactly one state at time t.

This is convenient, because we have: $p^t = Ap^{t-1}$. This just says that the probability that I wind up in, eg., state 1 at time t is the sum of the probability that I'm in state i at time $t-1$ and move to state 1, for all possible i's.

Notice that this is more general than the diffusion processes we solved with convolution. This matrix multiplication is only equivalent to a convolution if the matrix is a band around the diagonal, with every row the same, but shifted.

This formulation makes it easy to study the asymptotic behavior of a Markov chain. It is just the limit of:

$$A(A((A(p^0))) = A^n(p^0)$$

Notice that because A is stochastic, the elements of p^t always sum to 1.

As an example, let's consider the simple Markov chain given by: $A = [.75.5; .25.5]$; Using Matlab, we can see that if we pick p randomly and apply A to it many times we get $[2/3, 1/3]$ as our answer. We can work out that this is a steady state of the system. Suppose we have $p^t = [2/31/3]$. Then the chances that we will wind up in state 1 at time t+1 is $2/3 * 3/4 + 1/3 * 1/2 = +1/6 = 2/3$.

We can also see, with Matlab, that this asymptotic state is an eigenvector of A. This is because this repeated multiplication is just the power method of computing the eigenvector associated with the largest eigenvalue of A. That is, if we keep multiplying p by A, the result converges to the leading eigenvector. This is true regardless of the initial condition of p.

Proof by example: Suppose $p = av_1 + bv_2$, where v_1 and v_2 are eigenvectors of A, with associated eigenvalues of λ_1 and λ_2. Then:

$$A(av_1 + bv_2) = A(av_1) + A(bv_2) = a\lambda_1 v_1 + b\lambda_2 v_2.$$

And similarly: $A^n(av_1 + bv_2) = a(\lambda_1)^n v_1 + b(\lambda_2^n)v_2$. As n goes to infinity, if λ_2 is a smaller than λ_1, it will become insignificant, and the larger eigenvalue dominates. The result converges to a vector in the direction of v_1. This also means that the leading eigenvalue of A must be 1; this happens because A is a stochastic matrix.

I've skipped some conditions needed to prove this. First, the stochastic process must be ergodic. This means that we can get to any state from any other state. Otherwise, we might start in one state and never be able to get to the leading eigenvector from it.

Second, the Markov chain must have a unique leading eigenvalue. Otherwise, the result could vary among linear combinations of the leading eigenvectors. Or the result might not converge, but could be periodic. For example, if

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

then $(\frac{1}{2}, \frac{1}{2})$ is an eigenvector, but the result can also oscillate. Third, the initial condition must be generic. If it has no component of the leading eigenvector in it, it can't converge to that.

So the leading eigenvector of the transition matrix gives us a probability distribution over the asymptotic state. This can also be interpreted as the expected fraction of time the system stays in each state, over time. This makes sense, since the probability that the system is in a state at time n is the expected amount of time it spends there at time n.

3 Linear Relaxation Labeling

Now we're going to look at relaxation labeling, which is a classic approach to solving various image interpretation problems. In particular, we'll look at linear relaxation labeling, which is closely related to Markov chains. By the way, R.L. was developed at the University of Maryland by Hummel, Zucker and Rosenfeld. To some extent it has been superseded now by Belief propagation, but it's still interesting, it introduces some ideas that are used in belief propagation, and it also tends to get reinvented, so it's good to know.

Relaxation labeling is a way to assign labels to different portions of an image. It can be used when there are multiple possible labels, but we'll keep things simple by considering the case where there is just one, binary label. As an example, suppose we have a bunch of edgelets, and we want

to assign each one a label as either figure or background. We use relaxation labeling when the correct label for one image object provides information about other image objects. We iterate, and at each iteration we update the label of each object based on input from other objects. For example, if we have nearby, smoothly connecting edgelets, than if one is figure, the other is also likely to be figure.

In L.R.L. we assign each edgelet a measure of figureness that ranges from 0 to 1. 0 means it seems most like background, and 1 means most like foreground. Then we assign a weight from one edgelet to another. This weight is high if when one is figure, this makes us think the other is figure. To use this in updating the edgelet, we multiply the figure label of the other edgelet by this weight. We do this for all edgelets, and add up the results. So we can list the labels in a vector, v, and stick the affinities in a matrix. Then, to update the labels, we just have to multiply the label vectors by the affinity matrix.

Of course, this looks just like finding the asymptotic state of a Markov model. So one way to interpret Linear Relaxation Labeling is that a particle is jumping around from one edgelet to another. We start with particles distributed around the edgelets, and let them jump from one edgelet to another. They will tend to collect on the figure, where there is good continuation, and die out at isolated edgelets.

4 Intelligent Scissors

When you have a Markov chain, you can often solve problems using Dynamic Programming or even shortest path algorithms. A good example of this is with the intelligent scissors program. This is a program for interactive image segmentation. You start by clicking on one point on the boundary of an object. Then, as you move the cursor, the program interactively shows the ”best” contour from the first click point to the cursor. When you like the current path, you click again, nailing down that part of the path, and move on.

So the core problem is to find the ”best” contour connecting two image points. I’ll explain how this is done in a kind of stripped down, version of the Intelligent Scissors (IS) paper.

Suppose we want to find the best path from pixel p to pixel q. We build a graph in which each pixel is a node, and neighboring pixels are connected by an edge. Then we assign a cost to each edge, and run a shortest path algorithm to find the best path from p to q. Notice that with Equation (4), when we have a Markov chain, the - log probability is just the summation of a bunch of local costs based on local properties of the curve (curvature squared). So we will do that here. The energy of a curve, though, measured it’s likelihood in the absence of image information. In a real image, the gradient gives information about where the boundaries are, so we use that too.

First, curvature. Equation (4) suggests we should penalize paths with higher squared curvature. One problem here, though, is that we cannot measure curvature based just on two points in a path, while an edge in our graph is just between two pixels. The IS handles this by considering the direction of the image gradient at each point. Let p_i and p_j be adjacent pixels, with edge e_{ij} between them. Then we can base a curvature cost on $\nabla I(p_i)$ and $\nabla I(p_j)$. A simple such cost is:

$$\arccos\left(\frac{\nabla I(p_i)}{\|\nabla I(p_i)\|} \cdot \frac{\nabla I(p_j)}{\|\nabla I(p_j)\|}\right)^2$$

which is the change in angle squared. IS does something a little more complicated, taking into account the direction of $p_j - p_i$ also. It is also possible to represent paths using a node for each

position and direction, so an edge between two nodes can have a cost based explicitly on the path direction at each node (this is done, for example, in Shashua and Ullman's work).

In IS the edge cost is also based on two other terms. First, the magnitude of the gradient orthogonal to the path. That is, we compute the gradient at the location halfway between p_i and p_j, in the direction orthogonal to $p_j - p_i$. Second an edge detector is run on the image, and paths have lower cost if they go near edges.

While this is presented somewhat heuristically in the paper, this approach (and many related contour models) can be justified probabilistically. One could say that given two endpoints of a contour, we want to determine the maximum likelihood contour that connects them (or a probability that each pixel is on the contour) given a prior on contour shape and image information that locally tells us the probability that a position and orientation belongs to a contour.