Image Segmentation CS828 Spring '12

Angjoo Kanazawa

March 26, 2012

1 January 25th 2012 - Lecture 1

Definition: Segmentation (for this class):

- About low level vision in general
- Requires a lot of knowledge about th eworld, high level understanding, quite challenging.
- So we're going to focus on simpler segmentation that doesn't require that much knowledge about the world: Uniform surfaces, smooth shape. Still there will be varietion in intensity.
- Want to find uniform region in things (texture, color, motion, smoothness), not necessarily world property. Removed from true segmentation of objects but still useful.
- Image is an 2D geometric structure. Segmentation is clustering that takes advantage of this structure. Based on the assumption that near-by pixels have the same intensity.
-

We're going to look at

1. Diffusion
2. Anisotropic diffusion
3. Graph based algorithms: message passing, thinking of an image as a graph, every pixel is a node in a graph, edges to neighbors → Markov Random FIeld. Gives us a probablistic way to express the state of a node in relation to its neighbors. Usually NP-hard, but graph-cut and belief propagation algorithms still work. The biggest issue is when the number of labels is big.
4. Conditional Random Fields, a general version of MRF
5. Normalized Cut: form a graph
6. Wavelets

Math	Fourier transforms	Convolution	Diffusion
	Wavelets	Level sets	Riemannian Geometry

Current Research	Bilateral filtering (by Morel)	Texture Segmentation
	Cosegmentation	Affinity propagation

Workload

1. Reports (6 out of 8 papers):Be critical when reading papers, even if the paper is good, what is the really important. Learn to recognize, have a taste. (10%)
2. Presentations: 3 presentations per day, 15 min per paper 10 min each to discuss paper (15%)
3. a take home midterm, Final all on lecture material (50%)
4. Problem set/Project (25%)

2 January 30th - Lecture 2

2.1 Perceptual Grouping

- Putting pieces to preceive as a whole.
- Depends on the prior knowledge/statistics about the world.

History

- Behaviorists dominated in early 20th century, wanted to make psychology scientific, focused on quantifyiable things.
- Rejected anything introspective or mind building internal representations.
- AI, computers, chomsky killed behaviorists.
- Gestalt movement claimed visual system perceived world as a objects and surfaces, as a whole and not as raw atomic stimulus/intensities.

Classical principles/cues

- Knowing the role of edges is critical to how we perceive an image
- Similarity, Good continuation, Common Form, Connectivity, Symmetry (seems to jump out), Convexity, Closure, Common Fate, Paraallelism, Collinearity
- convexity beats symmetry? Connectivity also beats symmetry?

Theories

- We perceive shapes that are “good form”: smooth curves,, pretty abstract
- Bayesian: organizaton that’s most likely to be true. Not computationally friendly. Rather than checking all possible options, maybe we look for a certain small set of possiblities. Still doesn’t explain everything
-

3 February 1st - Lecture 3: Fourier Transform

3.1 Mathematical representation

a point in a $\mathbf{R^2}$ can be represented in a coordinate. If $p = (7, 3)$, we really mean $p = 7(1, 0) + 3(0, 1)$. *Any* point can be represented by a linear combination of two vectors. The basis vectors are:

1. Span the entire space: every point in the space can be written by linear combinations of these vectors.
2. Orthogonal: If not, moving in one direction will mean you'll be moving in the another direction
3. Unit: if not, the distance from the origin will not be constant.

We can compute the bases by

1. Linear Projection (inner product with each basis)

$$p = (p \cdot (1, 0))(1, 0) + (p \cdot (0, 1))(0, 1) \tag{1}$$

2. Magnitude of a point $||p||^2 = x^2 + y^2$ v

3.2 Functions in $\mathbf{R}^1$

The domain of the function is $[0, 2\pi]$, and we'll deal with functions in $\mathbf{R}^1$.

Def: a delta function:

$$\delta_s(t) = \begin{cases} 0 & s \neq t \\ \infty & s = t \end{cases}, \int_0^{2\pi} \delta_s(t)dt = 1$$

We'll write functions by using delta functions as a basis.

In infinite dimensions,

$$f(t) = \int f_s(\delta_s(t))ds$$

is the same as (1) but in infinite dimensions. Tw basis are orthogonal if their inner products are 0, in infinite dimensions, this is taking the integral. So delta functions are orthogonal.

This is a bad representation in some ways. It doesn't converge to the right representation (the function) quickly: using countable number of delta functions will not be a good representation of the function because it will only be correct in those places. We also need a lot of co-efficients.

Differen Representation Divide the interval $[0, 2\pi]$ into short k intervals with width $\frac{2\pi}{k}$. Use a rectangle in a interval as basis. They are orthogonal, so we can scale these rectangles and set it to a height that is equal to the average of the function in that interval. We have a piece-wise representation of a function using a finite basis. As $k \to \infty$, the approximation gets better. The *Reimann integral.* Here, we're stuck with a cetain level of accuracy as we fix k.

To get an arbitrary accuracy, we can reuse basis from multiple ks. i.e. if we divide the interval in 2, then 4, etc, then we'll get many rectangles or infinite bases that are *not* orthogonal, but can represent any function with finite pieces.

Functions are uncountable, but we're trying to represent it as a countable set of bases. But this is okay because we enforce the functions to be continuous.

3.3 Fourier Series

The basis elements:

- Height of $\sqrt{\frac{1}{2\pi}}$
- $\frac{\cos(t)}{\sqrt{n}}$ all are multiplied by a constant so when integrated it is 1.
- $\frac{\sin(t)}{\sqrt{n}}$
- $\cos(2t)$, $\sin(2t)$

They are unit vectors (normalized) and they are orthonormal i.e. $\int \sin(t)\cos(t)dt = 0$. But better, draw them around π. sin is symmetric around π, cos is negative symmetric. So if they are multiplied together, the signs are different so they cancel and gives you 0.

Now, we can write any function as an infinite sum of these basis elements:

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos kt + \sum_{k=1}^{\infty} b_k \sin kt \tag{2}$$

If the sums were finite upto N, then $\lim N \to \infty ||f(t)|| = 0$. This is a better representation then the delta functions because if we use enough co-efficients we will get really good approximation to the function.

$\cos^{2n}(t/2)$: Look at waht $\cos(t/2)$ look like, then raise it to a higher power. Really quicly, it will peak and look more like a delta function. By adding a constant in, $\cos(t/2 + a)$, we can shift the peaks.

Becasue we know that we can approximate any function with infinite delta functions, this means we can also do it with these basis. There are couple of identities by trigonometry to write higher power trig functions as a single power functions. i.e. trig functions with different frequences: $sin^2(t/2) = \frac{1-\cos(t)}{2}$, $sin^2(t) = \frac{1-\cos 2t}{2}$

Intuition: In practice, functions are smooth and with very small coefficients we can get a very good approximations.

Notation

$$\cos kt + i \sin kt = e^{ikt} \tag{3}$$

There are simple ways of computing these coefficients a_k, b_k. If we want a_k, we **take the inner product** of the function and $\cos kt$ i.e. $\int f(t) \cos ktdt$.

Complex case Given

$$c_k = \langle f, e^{ikt}, \rangle = \langle f, \cos kt \rangle + i \langle f, \sin kt \rangle,$$

$c_{-k} = \langle f, e^{i-kt}, \rangle = \langle f, \cos kt \rangle - i \langle f, \sin kt \rangle$

Then

$$c_k e^{ikt} + c_{-k} e^{-ikt} = a_k \cos kt + b_k \sin kt \tag{4}$$

We get back to the fourier representation.

Following from $a \sin t + b \cos t = c \cos(t + k)$, k is the phase, or the shift of functions.

Parsevaal's Theorem: Same as the pythagorean theorem:

$$\int f^2(t)dt = \frac{\pi}{2} a_0^2 + \pi \sum (a_k^2 + b_k^2)$$

This is good to use to measure how good our approximation is. So We can do

$$||(\int f(t) - a_0 - \sum_{k=1}^{N} a_k \cos kt - \sum_{k=1}^{N} b_k)^2|| = ||(\sum_{N+1}^{\infty} a_k \cos kt - \sum_{N+1}^{\infty} b_k)^2||$$

3.4 Fourier Transform

Let $f(t)$ is periodic going from $[0, 2\pi l]$. Then, we can represent $f(t)$ by

$$f(t) = \sum c_k e^{ikt/l}$$

(By dividing with l, we're stretching the basis element in $[0, 2\pi]$.) As $l \to \infty$, this gives us every possible fraction, all of $\mathbf{Q}$. Which mean we write this as:

$$f(t) = \int_{-\infty}^{\infty} F(k) e^{ikt} dk \tag{5}$$

Remember: e^{ikt} carries the orthonromal basis, now extending to all of $\mathbf{R}$, this means the coefficients are now in the ∞ domain so we write coefficients as $F(k)$, and call this the **Fourier transform** of $f(k)$.

(5) is the approximation of $f(t)$, the inverse operation to get the fourier transform is;

$$F(k) = \int_{-\infty}^{\infty} f(k) e^{-ikt} dk \tag{6}$$

e^{-ikt} is negative because it's the complex conjugate of e^{ikt}, (square it we multiply it with the complex conjugate.)

4 February 6th - Lecture 4: Smoothing & Convolution

Why do we *smooth* images? It's a way of passing information around, also it connects it more to segmentation (looking for a uniform property). When we smooth, we can take things that are similar and make them more similar. It also allows us to represent images in multiple scales, it helps us get rid of fine details, giving us coarser representations of an image. That is we want to remove high frequency portion and analyze the low frequency part.

Smoothing can be done by *convolution.*

In vision we always assume vision. Given a noisy input, the true intensity + noise, say $P_i = 100 + n_i$, smoothing takes the average of all pixels, we'll have

$$\begin{aligned} &= \frac{1}{M}\sum P_i \\ &= \frac{1}{M}\sum 100 + n_i \\ &= 100 + \frac{1}{M}\sum n_i \end{aligned}$$

A simple example of smoothing, the average of a lot of random variables makes the std of noise smaller.

4.1 1-D image

Think of 1-D images as a function: $f(t)$. We want to replace a pixel by the average of its neighbors. We write this as:

$$h(t) = \frac{1}{2\delta}\int_{t-\delta}^{t+\delta} f(t^{'})dt^{'} \tag{7}$$

(Where $t^{'}$ is just another points, not derivatives)

Let us define,

$$g(t) = \begin{cases} \frac{1}{2\delta} & \text{for } -\delta \leqslant t \leqslant \delta \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Note that $g(t)$ sums to 1. (Like a pdf of $U(-\delta, \delta)$.)

Now, we can write (7) as

$$h(t) = \int_{-\infty}^{\infty} f(t-u)g(u)du \tag{9}$$

It's as if we take the function g and winding it up so that it's centered around t and taking the inner product to get h. It flips, the left side of the filter is applied to the right side of point t. The resulted h is just shifting g at everypoint and taking the inner product.

We write this as

$$h(t) = g(t) * f(t) \tag{10}$$

It's natural to take the weighted average of your neighbors, because it's more likely that people around you have more information. So we use a gaussian filter.

4.2 Convolution

Two properties:

1. Linear: $g * kf = k(g * f)$, $g * (f_1 + f_2) = g * f_1 + g * f_2$

2. Shit-invariant: Take my function and translate, then convolve with a filter is same as convolving with a filter and shifting it. i.e. if $f'(t) = f(t+k)$, $h = g * f$, and $h^{'} = g * f^{'}$, then $h^{'} = h(t+k)$

Convolution Theorem Given F as the fourier transform of f, (F is the function of frequency), and the same for G, g, and H, h.

$$f * g = h \Leftrightarrow FG = H \tag{11}$$

Proof: $f * g = \int f(t-u)g(u)du$, call this h. To take the fourier transform of this, we take the inner product of h and e^{-itw}. So

$$H(w) = \int e^{-itw}(\int f(t-u)g(u)du)dt$$

Define $v = t - u$. Now,

$$\begin{aligned} H(w) &= \int e^{-itw}(\int f(t-u)g(u)du)dt \\ &= \int e^{-i(v+u)w}(\int f(v)g(u)du)dv \\ &= \int e^{-iuw}g(u)du \int e^{-ivw}f(v)dv \\ &= GF \end{aligned}$$

The sines and cosines are eigenvectors of functions because when you convolve it with any filter it just scales it.

The narrower the gaussian, the broader the fourier transform, The broader my gaussian, the narrower my fourier transform. So the lower frequency part gets preserved and the higher frequency (the edge of gaussian) gets reduced more.

Intuition: If the gaussian is so sharp that it's like a delta function, it'll only scale the function at that point t. Then, the fourier transform of a delta function is uniformly 1 at all frequencies. Because convolving with a delta function doesn't change anything, so $f * \delta = f$, bug $F * G = H = F$, so G is a uniformly 1 that doesn't change anything.

Similarly, if the gaussian is so broad that it's like a uniform function, then the fourier transform is like a delta function.

example A sinc function: $G(w) = \int_{-\delta}^{\delta} \frac{1}{2\delta} e^{-iwt}dt = \frac{2\sin(\delta w}{w}$. Plot it, bad fourier transform because the high frequency components go in and out. Compared, the gaussian filter provides us a very good fourier transform.

Why remove higher frequency components? If we assume the noise is i.i.d., we can show that the fourier transform of the noise is uniform. i.i.d. noise has equal energy. Bc this noise has the same energy everywhere, it's called the *white noise.* If we think of our image as some smooth pixels with a white noise. Images tend to have much more low frequencies than high frequencies. Noise is equal in low and high frequency, so if you reduce the high frequency components, it significantly reduces the noise.

Band-pass filter: Looks like $U(a, b)$, it perfectly preserves the low frequency component. It's fourier transform is the sinc function. So there's limitation to use these perfect filters.

High-pass filter: inverse of $U(a, b)$.

Fourier series:

$$f(t) = a_0 + \sum a_k \cos(kt) + \sum b_k \sin(kt)$$

(K is the frequency) The derivative:

$$f'(t) = \sum -a_k \sin(kt) + \sum b_k \cos(kt)$$

This is also a fourier series, *taking the derivative has the effect of scaling the coefficients by the frequency.* As frequency gets higher, it amplifies the coefficients.

This is why it's dangerous to take the derivative of a noisy image, because the derivative ampliflies the high frequency components with a lot of noise.

Taking the derivative is like convolution.

Gaussian Filter For any function, the more spatially localized (peaked) it is, the broader it is in frequency. Vice versa.

5 February 8th - Lecture 5: Diffusion

Sampling theorem Given

$$f(t) = a_0 + \sum_{k=1}^{T} a_k \cos(kt) + \sum_{k=1}^{T} b_k \sin(kt) \text{ for } t = 0, \frac{2\pi}{M}, 2\frac{2\pi}{M}, 3\frac{2\pi}{M}$$

This equation is linear in unknown coefficients ($2T + 1$ many) so we need $2T + 1$ samples to solve this equation.

If we have an analog version of someone's speech, you can digitize it by taking $2T + 1$ samples knowing they are band-limited (otherwise we'll lose information). Speech is fine but the best thing is to apply a band-pass filter to make sure that it's band-limited.

If the signal isn't band-limited to begin with, i.e. $f(t) = a_0 + \sum_{k=1}^{3T} a_k \cos(kt) + \sum_{k=1}^{3T} b_k \sin(kt)$, you have more unknowns with $2T + 1$ samples, and you're ignoring a lot of information and it's totally meaning less.

Aliasing when high frequency is indistinguishable from low frequency when sampled.

5.1 Diffusion

Diffusing is like smoothing, provides a physical analogy to smoothing. By setting it up and solving diffusion as a PDE, we'll see that writing smoothing as PDE can be modified so that edges can be preserved?

Again, everything followed is in 1D, we go from discrete, continuous, then back to discrete.

Discrete Imagine you have a lot of small buckets with lots of particles in it, describe each bucket by how many things are in it, the *concentration*, $C(x, t)$, which changes over discrete time steps. i.e. $C(1, 0)$ tells us how many particles are in bucket 1 at time 0. Some of these particles can jump to neighboring buckets. A reasonable model for physical diffusion (milk and coffee, heat, etc). *Flux* , $J(x, t)$, is the net number of particles that are moving in the positive direction.

Diffusion is *isotropic* (equally likely to go to left and right) and *homogenous* (same things happen everywhere).

The relationship between flux and concentration can be modelled as such:

$$J(x, t) = -D\frac{\partial C}{\partial x} \tag{12}$$

If more stuff goes to the left than the right, the flux is negative. D is a constant diffusion coefficient, what fraction of things move around, the "diffusivity" of particles. If D is low, less stuff moves around.

$$\frac{\partial C}{\partial t} = -\frac{\partial J}{\partial x} \tag{13}$$

How much C changes over time? Then I want to count how much is coming in and how much is coming out (flux). So if flux is constant, then the concentration is not changing. If the flux is increasing, that means there's more stuff going out to the right. So if the change of J is positive, the change in C is negative.

If D wasn't constant, it would depend on x, to do more interesting kind of smoothing, we can make D into a function of x.

Combine the equation to get rid of J by taking PDE wrt x:

$$\frac{\partial J}{\partial x} = -D\frac{\partial^2 C}{\partial x^2}$$

Plugging this back to (13), we get

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} \tag{14}$$

A positive second derivative means concavity, a local minima, so the concentration increases, similarly, concentration goes down at a local maxima second derivative negative. This means that concentration is being smoothed over time.

Numerical Analysis A finite differential problem. Taking the taylor series for a fixed t, we get

$$c_{i+1} = c_i + \delta x\frac{\partial C}{\partial x} + \frac{1}{2}\delta x^2\frac{\partial^2 C}{\partial x^2} + \mathcal{O}(\delta x^3) \tag{15}$$

(also $c_{i-1} = c_i - \delta x\frac{\partial C}{\partial x} + \frac{1}{2}\delta x^2\frac{\partial^2 C}{\partial x^2} + \mathcal{O}(\delta x^3)$)

Ignoring the higher order terms, you get, in first-order,

$$\frac{\partial C}{\partial x} = \begin{cases} \frac{c_{i+1}-c_i}{\delta x} \\ \frac{c_i-c_{i-1}}{\delta x} \end{cases} \tag{16}$$

Adding them together, we get

$$\frac{\partial C}{\partial x} = \frac{c_{i+1}-c_{i-1}}{2\delta x} \tag{17}$$

Better because this is symmetric.

Doing the same thing to the second derivative (difference between the first derivative of left and right)

$$\frac{\partial^2 C}{\partial x^2} = \frac{(c_{i+1}-c_i)-(c_i-c_{i-1})}{\delta x^2} \tag{18}$$

We could say similar thing to wrt to t:

$$\frac{\partial C}{\partial t} = \frac{c(x,t+1)-c(x,t)}{\delta t} \tag{19}$$

Putting all of this together, (14) becomes

$$\begin{aligned} \frac{\partial C}{\partial t} &= D\frac{\partial^2 C}{\partial x^2} \\ \frac{C(i,t_0+1)-C(i,t_0)}{\delta t} &= D\frac{C(i+1,t_0)-2C(i,t_0)+C(i-1,t_0)}{\delta x^2} \\ C(i,t_0+1) &= C(i,t_0) + \frac{\delta tD}{\delta x^2}(C(i+1,t_0)-2C(i,t_0)+C(i-1,t_0)) \\ &= (1-2\lambda)C(i,t_0) + \lambda(i+1,t_0) + \lambda C(i-1,t_0) \end{aligned}$$

Where $\lambda = \frac{\delta tD}{\delta x^2}$. This is just another convolution with a filter that looks like $l = (\lambda, 1-2\lambda, \lambda)$.

Let $C(x,0) = f(x)$, to get the concentration at time n, I get the initial concentration at time 0 and apply convolusion with filter l t_0 times. i.e.

$$C(x,t_0) = (l \times l \times \cdots \times l) \times f$$

But since convolution is associative, we can combine the filters together. Convolving l with l over and over gives us a gaussian.

So *diffusion is just a convolution with gaussian, same thing as low-pass filtering an image, smoothing.*

Limitation on λ:

$$\begin{aligned}1 - 2\lambda &> 0\\ 1 - 2\frac{\delta t D}{\delta x^2} &> 0\\ \frac{\delta x^2}{2D} &> \delta t\end{aligned}$$

Another intuition Consider a single particle that is diffusion¡ This is a random variable x_i, where $x_i = -\delta x$ if it moves left at time i, δx if it moves to right. After T time steps, the position of the particle is $\sum_{i=1}^{T} x_i$, where by LLN, this is a r.v. with 0 mean Gaussian distribution. So the particles position after T time steps is a Gaussian, we can get it by convolving x_0 with a Gaussian or convolving it with a filter T times, same thing.

6 February 13th - Lecture 6: Edge Detection

Paper Presentation Topics

- Graph based, MRF/CRF
- Texture: (texton-boost)
- **Co-segmentation**
- **Layout** (3D surface estimation) by Hoeim, Efros
- Affinity propagation by Frey (tronto)
- Edge detection: Malik, Basiri
- Graph Cuts - Galun 'Detecting and Sketching the Common"
- Semantic

6.1 Edge Detection: Canny Edge Detector

Basic Idea: look at sudden changes in intensity and the first derivative I_x. But we'd expect some noise, so we always have to smooth the image with a gaussian before we take the derivative.

Algorithm in 1D:

1. Smooth with a gaussian
2. Take the first derivative
3. (a) Is it strong? (of large magnitude) Everything above a certain threshold is strong, where image is changing rapidly.
   (b) Pick points that are not only strong but also a local extrema

This procedure is optimal to minimize the number of false detections, while also optimizing how well we localize the edge.

Couple of parameters: The threshold is the tradeoff between false positive/negativees, std of the Gaussian, σ, is the width of the filter, the wider it is the smoother I and less noise, but less accurately we'll localize the edge.

6.2 In 2D

In 2D, things are little bit more complicated. A rapid intensity change depends on the direction. We need to figure out which direction the intensity changes most rapidly.

Compute the image gradient, (I_x, I_y), and the magnitude $||(I_x, I_y)|| = \sqrt{I_x^2 + I_y^2}$. We can represent the direction with the maximal change by a unit vector:

$$\frac{(I_x, I_y)}{||(I_x, I_y)||} \tag{20}$$

Intuition: When you take the first derivatie, you assume the image is locally linear, like a tilted plane, where there's a direction with big change, where orthogonal to that direction change is flat.

We take gradient ∇I instead of a derivative in 2D, and in asking is it strong, we ask if $||\nabla I||$ big. It's bad if you don't use a big enough gaussian. We want to renormalize so that the filter sums to 1, otherwise it makes the image dimmer or lighter than it actually is.

Smoothing in 2D
Gaussian in 1D:

$$\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}}$$

Gaussian in 2D:

$$\begin{aligned} G(x,y) &= \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{x^2+y^2}{2\sigma^2}} \\ &= \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}}e^{-\frac{y^2}{2\sigma^2}} \end{aligned}$$

(Maybe midterm: prove that these are same things) We can use seperable filters because we can divide the filter into 2 filters.

Picking extrema in 2D What defined an edge is the local extrema *in the direction of the gradient.* The orthogonal direction is where the edge continues. A subtlety: Given a vector field, every pixel has a vector which is its gradient. We want to know what the image gradient is where each vector is pointing, but there might not be any pixel ther. We can calculate what it would be if the pixel grid were dense by interpolating (linear or bi-linear) between two image locations.

Hysteresis to figure out if $||\nabla I||$ is big: if i pick a threshold that's too big, things might get fragmented and miss some edges. With a lower threshold, we'll get unneccessarily edges from noise in the background. We want the best of both. One heuristic to get this is to do the high threshold, then the low threshold to get weaker edges but only keep them if they're close to the stronger edges. This is a very basic perceptual grouping based on connectivity.

This is the prominent method for edge detection but it still doesn't really work in real images. No matter how you pick threshold, we get too much or too less. Because edges/boundaries that are intuitive to us may not be strong locally. Also around pointy edges/corners, smoothing weakens them.

6.3 Corner Detection

A way to define a corner is a smal region in the image where you have image gradient change in both directions. Look at a small window (5 x 5), in this window look at the image gradients (25). Do PCA, principal component analysis, on the gradients and find out how much variations there are in image gradients in one direction. If image gradient only goes in one direction, only one PCA will be strong, but if it goes in x and y direction, both PCA will be strong.

Compute a scatter matrix,

$$H = \begin{pmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_y I_x & \sum I_y I_y \end{pmatrix}$$

Where the first eigen value λ_2, smallest one, gives you how much gradient is in the direction of least. if both λ_2, λ_1 big, it means gradients change in all direction. This is call the harris corner detector.

7 February 15th - Lecture 7: Non-linear Diffusion

Presentation points:

- Understand what's the biggest contribution about the paper. No need to pay attention to all of th edetails in the paper.
- Give context of this paper from state-of-the-art, past, and how it fits
- Where significant problems identified, addressed?
- Give opinions
- Always understand the questions before answering it

7.1 Nonlinear Diffusion

Goal for today Non-homogenous diffusion: when you're near the boundary don't smooth so much. Anisotropic (# of particles leaving is not same in all direction):.

In 2D isotropic, the flux is in the direction of the negative gradient, down hill. In anisotropic, the matrial isn't necessarily going down hill, the direction depends on a larger context.

7.2 Review: Isotropic Diffusion

$f(x)$ is the image at time 0, and $u(x,t)$ is the image at time t Flux in 1D:

$$j(x,t) = -D\frac{\partial u}{\partial x}$$

flows to the negative direction of the derivative. D is how fast stuff diffuse. Flux in 2D:

$$j = -D\nabla u$$

If D is constant, this is gaussian smoothing in 2D, isotropic and homogenous. If we make $D = D(x)$ a function of location x, it's non-homogenous. D could also be a tensor, a 2x2 matrix, giving us two vectors, now diffusion is non-homogenous and anisotropic. Intuition in 1D:

$$\frac{\partial u}{\partial t} = -\frac{\partial j}{\partial x}$$

in 2D:

$$\frac{\partial u}{\partial t} = -\text{div} j$$

Where

$$\text{div} j = \frac{\partial j}{\partial x} + \frac{\partial j}{\partial y}$$

Substituting things, the heat equation in 1D

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$

in 2D

$$\frac{\partial u}{\partial t} = \text{div}(D\nabla u)$$

Gradient is a vector $\nabla u = (\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y})$. div$j$ is the sum of these, saying how much material is piling up in this location.

7.3 2D Non-homogenous, isotropic diffusion

Let $D = g(x)$, a function of the image. The amount of diffusion is different in locations but the way things diffuse is the same. Write $\frac{\partial u}{\partial t} = \text{div}(g(x)\nabla u)$.

We want to diffuse but not around the boundary, i.e. when the gradient is big, not. To do this, we can make g:

$$g(||\nabla f||^2) = \frac{1}{\sqrt{1 + \frac{||\nabla f||^2}{\lambda^2}}}$$

When $||\nabla f|| \to \infty$, $\sqrt{1 + \frac{||\nabla f||^2}{\lambda^2}} \to \frac{\lambda}{||\nabla f||}$, very small. So we'll smooth less and less as the gradient gets bigger. λ is set by hand, controls what's the point where we make things non-homogenous (where to stop diffusing). This gives a linear PDE, and we can't do this with a convolution anymore (because convolution applies same thing everywhere).

Problem with this approach is that it depends on the original image. Say we have a small gradient that gets smoothed out. It'll have less influence but small structures like this leave artifacts. Instead, we can take the gradient of the actual image (not the original f).

$$g(||\nabla f||^2) = \frac{1}{\sqrt{1 + \frac{||\nabla u||^2}{\lambda^2}}} \tag{21}$$

This is the **Perona-Malik** model. This model is *unstable.* Why? When a gradient really big at one point, its left neighbor is not getting anything, so the neighbor goes down (with a slow bumpy slope, we can get staircases) i.e. things are going in the opposite direction than usual diffusion (instead of hi to low, it can go from low to hi). So this is unstable. We can go around this by smoothing the image before taking the gradient.

7.4 2D Non-homogenous, anisotropic diffusion

Let D a 2x2 matrix based on the image. So back to $\frac{\partial u}{\partial t} = \text{div}(D\nabla u)$. D has eigenvectors v_1, v_2, v_1, where $v_1 \parallel \nabla u_\sigma$ and $v_2 \perp \nabla u_\sigma$. Set the eigenvalues λ_1, λ_2 to $\lambda_1 = g(||\nabla u_\sigma||^2)$ (the term in Perona-Malik) and $\lambda_2 = 1$.

If $\nabla u_\sigma = \nabla u$, we're just scaling λ_1 by sometime, and this is just Perona-Malik. u_σ is the original image smoothed by gaussian of size σ.

Intuition: around the boundary, at a finer scale, the gradient is showing noise. If ∇u_σ large, means that we're near a boundary. ∇u_σ has the largest direction v_1 (pointing towards the boundary), and a component orthogonal v_2. We can always take ∇u and decompose it into v_1, v_2. If ∇u_σ big, v_1 gets scaled down, but v_2 is totally preserved (it's just 1), so ∇u *points away from the boundary.*

These approaches are all still heuristics compared to the bilateral filters.

8 Lecture 8 - Contours

8.1 Markov Representation

Each variable depends on its neighbors. Conditionally independent of the variables far away.

Markov Chain: R.V. occuring over time. 1D. Markov Random Field: 2D. In an image, a markov field, a pixel depends on the immediate neighbors, but conditionally indepenent of the rest of the pixels given the neighboring pixels.

In 1D, key is that if you know a single r.v. it divides the set into two disjoint sets. The seperation allows us to use DP etc to find optimal solution. In 2D, a pixel doesn't divide the image in two disconnected sets, so the problems with optimal solution in 1D becomes NP-hard.

Sometimes we want to have a stochastic model of images/objects. Deniosing is a random process that adds noise. To get the MAP estimate, I need some expectation about the image, some prior. We can model the boundaries of objects, having some prior model of the shape of an object is important, and markov models help us build these prior.

One way to describe a texture is repetition with variation or samples drawn from some random process.

1D case Markov chains is important for classification: positive examples are similar meaning that they are samples from the same distribution. We can model actions/image sequences in a markov chain.

Definition: $x_1, x_2, \dots, x_n$ are random variables. x_is form a Markov Chain iff.

$$P(x_j|x_{j-1}, x_{j-2}, \dots, x_1) = P(x_j|x_{j-1})$$

Diffusion is an example of a markov chain. Knowing where the paritcle was at time $t-1$ helps us guess where the particle maybe at time t, but knowing that at time $t-2$ doesn't really help.

Markov chain that can model contours. You want to find a contour in the image that fits the image gradients (the edges) but it's also plausible (to be a real shape). We want to solve the contour, c, given the image I. This is

$$\arg\max_c P(c|I) = \frac{P(I|c)P(c)}{P(I)} \approx P(I|c)P(c)$$

(The denominator doesn't matter because $P(I)$ same for all. $PI(c)$ is is this contour plausible, is it likely to be a boundary of a real object? To do this we need a prior.

We can learn such distribution, but generate a simple model. Define contour to be a point that moves through a space with some particular direction. Denote the direction by θ. Then we can say

$$x' = \cos\theta, x' = \sin\theta, \theta' = \mathcal{N}(0, \sigma^2)$$

where x' is the change in the x-coordinate etc. If sigma is small, the countor is smooth (straight) and not very contorted. In another words, θ is going through a diffusion process and the direction follows from it..

Let Γ_t is the position of the contour at time t. Then, without constant,

$$P(\Gamma_t) \approx \prod e^{-\frac{\theta'(t)^2}{\sigma^2}}$$

Better written as (dropping constant):

$$-\log P(\Gamma_t) \approx \int \theta^2 dt \tag{22}$$

"Snakes" computes a contour and also acounts for the prior probability of the contour, minimizing (22).

Is this a really good prior for contours? Obvious fixable problems:

- Probability of generating an perpendicular edge (discontinuity) is 0. Fix by adding a gaussian mixture for the model that θ'
- Convex shapes are relatively likely, where curvature is all positive, meaning curvature is in a single direction. But this model allows θ' to go in both directions.
- We also want contours that are short and smooth. How: Fixed probability that contours will end, i.e. particles have a half life λ that we add to (22)

Curve of least energy, curve between two edges that minimizes (22) because it can be thought as a energy of contours.

9 Lecture - 9 Markov Random Field

Term project: Baseline-comparison of several algorithms, High end-conference paper submission. EC for doing both.

9.1 Markov Property

$x_1, \dots, x_k$ form a markov chain if $P(x_k|x_{k-1}, \dots, x_1) = P(x_k|x_{k-1})$. Assume x_k has a label, $x_k \in S = s_1, \dots, s_m$. Probability distribution for x_k, $P(x_k)$ is the probability it has each label. We can express this in a vector $P^K = \langle P(x_k = s_1) \dots P(x_k = s_m) \rangle$. Using the markov property, the transition can be represented in a matrix multiplication.

$$P(x_k|x_{k-1}, \dots, x_1) = \sum_{j=1}^{M} P^{k-1}(j) P(x_k = s_i | x_{k-1} = s_j) \tag{23}$$

Where we let $A_{ij} = P(x_k = s_i | x_{k-1} = s_j)$, an entry in the transition matrix $A.A_{ij}$ is going from state j to state i, always same over time. If columns of A sum to 1, we say that A is a stochastic matrix. Here, it should sum to 1 because a column is probability of going from j to all i. Then,

$$P^k = AP^{k-1} \tag{24}$$

The inner product of the first row of A and P^{k-1} is same as (23).

A is an easy way to figure out the stable distribution of this chain. i.e.

$$\begin{aligned} P^2 &= AP^1 \\ P^3 &= AP^2 = A(AP^1) \\ &\vdots \\ P^n &= A^{n-1}P^1 \end{aligned}$$

In the end we get the largest eigenvector of A. This is actually the power method of finding eigenvectors. Let $P^1 = a_1 v_1 + a_2 v_2$, where v_1, v_2 are the eigenvectors. If we apply A to P^1, we get:

$$\begin{aligned} AP^1 &= a_1 A v_1 + a_2 A v_2 \\ &= a_1 \lambda_1 v_1 + a_2 \lambda_2 v_2 \end{aligned}$$

And in general

$$A^n P^1 = a_1 \lambda_1^n v_1 + a_2 \lambda_2^n v_2 \tag{25}$$

For a stochastic matrix, largest eigenvalue is 1, so we preserve v_1. If we want to know the steady state of a markov chain, we take the eigenvector corresponding to the largest eigenvalue.

Technicality: P^1 matters to do this: It has to be possible to reach any state from any other state. And largest eigenvalue has to be unique.

Diffusion is a markov process. Recall: gaussian smoothing is repeatedly applying a filter like $\lambda, 1 - 2\lambda, \lambda$ to an image. i.e.

$$\begin{pmatrix} \lambda & 1-2\lambda & \lambda & 0 & \dots 0 \\ 0 & \lambda & 1-2\lambda & \lambda & \dots 0 \\ \vdots & & & & \end{pmatrix}$$

Convolution is equivalent to matrix multiplication with this matrix. Eigenvectors of this is harmonic, sines and cosines.

9.2 Relaxation Labeling

Basic idea: Suppose I want to label regions. If you know that there's a telephone, the thing underneath it is likely to be a table etc. The label of one thing gives you evidence about labels of other objects in the scene. The point of relaxation labeling is to figure out how to combine all these information.

Linear version of relaxation labeling assumes that evidences can be linearly combined. An example of edge labeling: label foreground or background. I.e. things that are close to each other and connect smoothly to each other are foreground. One way to solve this problem by prodviding edges between line segments. If the edge between segments is strong, then the two segments are highly correlated. Initialize everything at 0.5. Initial condition doesn't matter. This is different from belief propagation, because a node may get a strong evidence but it's not an independent evidence. (Because you started the rumor). In belief propgation, you're really careful not to double count an evidence.

Another interpretation of this algorithm is that edges are like transition probabilities of particles. Steady state is defined by how well particles can get to certain nodes.

1D case is nice because we can do matrix multiplication to find the steady state i.e. taking eigen values. You can also do dynammic programming.

9.3 Intelligent Scissors

An interactive segmentation method. Idea: click on two different pixels. Find the best boundaries that connect those two points. Because it's interactive, it doesn't have to be perfect. In order to solve this, we use markov/diffusion model for contours. We want the contours to have relatively low curvature (smooth) and close to points of high image gradient.

Add edges and weight it with whether this edge is a good boundary or not, then just use any shortest path algorithms (DP). Take the perpendicular derivative of an edge, you get high cost if derivative is small, vise versa. Cost is high if you're not following the boundary (gradient).

Next thing is smoothness/curvature. You need three points to define a curvature. Create a node for every pixel, find its image gradient, when you move, the penalty is if the image gradient is changing a lot or not.

9.4 Review of DP

Problem: we know the state as $x_1 = 0$, $x_n = 1$. Find the most likely sequence of intermediate states given the markov model, kind of like the contour problem. The most trivial solution is consider all possible combinations and try it - clearly exponential. You don't really have to test all. If some intermediate $x_k = 0$, we can split the probelm into two: what's the problem of going from x_1 to x_k, x_k to x_n, same for $x_k = 1$.

10 Lecture 10 - Markov Random Field

MRF is a collection of sites $S = \{s_1, \dots, s_n\}$ (in 2D grid of sites $S = \{S_{ij}\}$), and a set of labels $L = \{L_0\}$). Every site is labeled with a labeling $f = \{f_1, \dots, f_n\}$.

It would make sense to talk about $P(f)$, which is the joint probability of assigning a label to all sites. With a markov chain, s_k will be dependent on f_{k+1}, f_{k-1}, but given those it's conditionally independent from the rest. Here, similar, but not two but all the neighbors and a site is conditionally independent given all the neighbors, the markov blanket. Formally, the neighborhood of s_i is N_i, where $s_i \notin N_i$, and $s_i \in N_j \iff s_j \in N_i$. An MRF is $P(f) > 0 \forall f$. The key property is that

$$P(f_i | f_{S-\{i\}} = P(f_i | f_{N_i}) \tag{26}$$

You can take any probability distribution and make it in to a MRF just by making everything into neighbors. The computational effort depends on the size of the neighborhoods. If the neighborhood size is too big, we don't get the benefit from making it into a MRF. Think of a grid structure.

Example: in denoising, the label for each pixel is its true noise free intensity. Putting it in MRF says that the true intensity of a pixel depends on the true intensity of its immediate neigbhors but it's conditionally independent of the rest given those neighbors. In stereo, labels are the disparity level. We can also use this for less structured (not grids) things, like detecting line segements and labeling it as a foreground/background, where the neighbors are defined by those segments around the line segment.

We can ask questions like "What's the MAP estimate of assigning labels?", marginalizing over one site/rv: "what's the probability distribution of that site given the rest?". We also want to learn this probability distribution modeled by MRF, i.e. if we get bunch of examples of labeled images. But these questinos are not trivial.

The big result that brings all of this together is the equivalence between MRF and the gibbs distribution.

10.1 Gibbs Distribution

A clique is a set of sites $\{s_{i_1}, s_{i_2}, \dots\}$ s.t. $s_{i_j} \in N_{i_k} \forall j, k$. $P(f)$ is a **gibbs distribution** iff.

$$P(f) = \frac{1}{Z} e^{-u(f)/T} \tag{27}$$

T is referred to as the temperature and $u(f)$ is the energy function/potential, can be written as

$$u(f) = \sum_{c \in C} V_c(f) \tag{28}$$

Where V is the clique potential of C. In a grid, we have a trivial clique (by itself), two clique (two pairs) in vertical and horizontal direction. In (27), the exponent should look like summing multiple probabilities of the clique. Z is the normalization factor $Z = \sum_{f \in F} e^{-u(f)/T}$.

For vision, MRF is builds a prior. But we haven't discussed about how well this prior fits to the image we have (i.e. denoising). So we have **observations** , X.

Usual vision problems form MRF s.t. every obesrvation have a specific independent structure. I.e. given $X = \{x_1, \dots, x_n\}$,

$$P(x_i | f, X / \{x_i\}) = P(x_i | f_i) \tag{29}$$

Implicitly we've had this assumption in the denoising papers we read, because we assume that the actual observation (noise+truth)'s noise is independent of every other pixel's noise.

Continuing with the denoising example, we want solve this labeling problem. In denoising it amounts to recovering the true intensity. So given an observation, $x_i = f_i + e_i$ where e_i is an i.i.d. noise from $\mathcal{N}(o, \sigma^2)$. We want

$$\arg\max_f P(f|X) = \arg\max_f \frac{P(X|f)P(f)}{P(X)}.$$

by Bayes law. Well

$$P(X|f) = \prod P(x_i|f_i)$$

because of (29), and in image denoising, we have $P(x_i|f_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(f_i-x_i)^2}{2\sigma^2}}$.

We can define the single clique potential as

$$V_c(f_i) = \frac{(f_i - x_i)^2}{2\sigma^2}$$

and for a pair clique, we define $V_c(f_i, f_j) = \begin{cases} 0 & \text{if } f_i = f_j \\ k & \text{otherwise} \end{cases}$

The mot probable configuration is where the whole image is constant. Total variation prime which is an explicit description would be $V_c^{tv} = (f_i, f_j) = |f_i - f_j|_l$, but practically people don't do this. And the optimal MAP estimates of MRF is $\mathcal{NP}$-hard

Once we have these clique potentials, we have an optimization problem of minimizing $u(f)$.

White noise is called white because the fourier transform of the noise is constant. Color noise might have a low freq components in fourier transform, which means the noise of the neighbors are related. What if we have color noise? If we know f_1, previously we would've said x_1 is independent of everything else. But now since the noise is correlated, it's not the case here because the observations are related now and we have a v-structure.

Classic example: earthquake and burglur. They are independent event, but if we give an observation that an alarm goes off. This gives evidence about the burglar and the earthquake and makes them dependent because if earthquake happened, there's a low chance that burglary also happened.

Now *condition everything on the image* .In MRF, we can figure out the joint distribution of everything and we have a full generative model $P(f, X)$. But if we condition, we get the **conditional random field** where we sought for $P(f|X)$. We don't have the generative model anymore, we don't have the distribution of the model, but having the image gets rid of the dependency problems.

11 Lecture 11 - CRF

SItes $= S = \{s_i\}$, labels $f = \{f_i\}$, Neighborhoods: $j \in N_i \iff s_j$ is a neighbor of s_i.

Two restrictions to be a MRF: $P(f) > 0 \forall f$ $P(f_i | f_{s-\{i\}} = P(f_i | f_{N_i})$

For **CRF**, the graph is globally conditioned on the observation. None of the sites will be conditionally independent because all of the sites share the same observation.

How does this change things?

11.1 Gibbs Distribution

Equivalent to MRF. For something to be a gibbs distribution, we say it has this form: $P(f) = \frac{1}{Z} e^{-u(f)/T}$ where $u(f)$ is the energy. Key things is that u can be factored. In particular, we say $u(f) = \sum_{c \in C} V_c(f)$. where C is the set of all of the cliques, and V_c is the potential of clique c.

In CRF, the only difference is that we have $u(f, x)$, energy depending on the labels AND the observations. Where

$$\begin{aligned} u(f,x) &= \sum_{c \in C} V_c(f, x) \\ &= \sum V_c(f_i, X) + \sum V_c(f_i, f_j, X) \end{aligned}$$

Pictorically, in MRF, every site has a single observation (value of pixel at i), in CRF, every site has a same observation, all of observed data. With a CRF, the label can be dependent on the whole image, or a piece of a image, but in MRF, you can only model the dependency on a single pixel.

MRF is generative, it models the entire probability distribution $\arg\max_f P(f|X) \propto P(X|f)P(f)$. This might be very difficult! CRF models $P(f|X)$ directly and it's a discriminative model.

11.2 MAP Inference

Problem of $\arg\max_f P(f|X)$. What's your best guess as to what's the best answer?

In general NP-hard, so all solutions are iterative. How to chose the intial point? Heuristic is: try a bunch of random solutions, iterating it until you hit a local solution, pick that as your starting point. Only helpful if you can get the right place to start in 10%, but in MRF usually it's so rare to hit the rigth place that it won't really help. Another approach is ignoring the interaction potential. Just base the starting point on the unary potential.

Iterated Conditional Modes $\hat{f}$ current labeling. Perform: for random i, $\hat{f}_i = \arg\min_{f_i} u(f_i \cup \hat{f}_{j \neq i}, X)$ (maximize the probability so minimize the energy). If unary constraint is really strong, this is going to find the optimal. This gets stuck really easily in local-minima.

Simulated Annealing On convex function: Start some place, you move down (towards the minima). If your function is not convex (with a lot of local minima), even if you're at a local minima you should be bouncing around to get out of it.

Given $\hat{f}$, pick f_i randomly

$$p = \min(1, \frac{P(\hat{f}_{i \neq j} \cup \{f_i\})}{P(\hat{f})}$$

if the ratio > 0, this means that the old labeling was better, with ICM, we would've never moved. Instead here we still move sometime.

T in the gibbs distribution is the temperature. $e^{-u(f)/T}$, when T is high, the probability landscape is very flattened and the ratio is around 1, so whether it's good or not we always move. As temperature gets low, the labeling with the highest probability dominates (gets 1 and the rest 0), and the more spread out the probabilities are and less likely for us to move again.

11.3 Graph Cuts

Simulated annealing is old, not ppl use graph cuts.

Intuition: With ICM, you're considering one node at a time and saying would it improve if I change this? Idea here is to consider changing a lot of nodes at the same time. Two steps: α-expansion and $\alpha - \beta$ swap.

Problem of ICM is that pairwise constraints will not like changing one pixel at a time. $\alpha - \beta$ swap can swap all the pixels that has either label, fix all other pixels, then relabel all those pixels in an optimal way just by using the two label. If this labeling is better, we'll take it. This is called the graph cut because we're making it into a graph and the min-cut corresponds to the best configuration of $\alpha - \beta$ labels.

$\alpha - \beta$ **swap** Construct a graph where the terminal nodes are α and β, take all sites currently labeled α or β as nodes and connected to both terminal nodes. We want the best cut where the cost of the cut is the weight of the edge. The cut corresponds to a labeling, and weight corresponds to the clique potential of that labeling. Goal is to find a cut so that α and β are disconnected. If you cut the edge connecting to α, that nodes gets α. Think of edge as the clique potential, so cutting the edge is like taking the potential.

The min-cut will always keeps exactly one of the edge connecting to a terminal node. You also make edges between the sites, so that if sites don't ahve the same label they need to be cut. To make sure that each edge represents the clique potential, for example a site's edge connected to α has weight $V_c(f_i = \alpha, X) + \sum_{j \in \mathcal{N}_i} V_c(f_i = \alpha, f_j, X)$ where $f_j \neq \alpha, \beta$. For edges between sites, the weight there is $V_c(f_i, f_j, X)$. For this to work, you need to design the pair-wise potential to be s.t. if $f_i = f_j$, cost or the potential is 0. It also has to be symmetric (doesn't matter if it's labeled α or β).

We create this graph, then do a min-cut $O(n^2)$ pretty fast practically.

α**-expansion** Similar, whether they keep α or everything else become α or stay the same.

The whole algorithm is start with an initial labeling, then do $\alpha - \beta$ swap (or α-expansion) until you converge.

Suppose we have a binary labeling problem. Then if we do min-cuts we have the globally optimal solution. So binary problems aren't really NP-hard.

12 Lecture 12 Normalized Cut

MIn cut - cost of the cut is the sum of the weight of the edges, the min cut is the cut with smallest cost.

Normalized Cut: to assign weight, you can look at the intensity differences $(I_i - I_j)$. But you want it s.t. weight is really low when the intensities are very different, so you do

$$\exp(-\frac{||I_j - I_j||^2}{\sigma}) \tag{30}$$

This is heavily biased towards smaller regions. If the edge weight is ϵ, but the image is big enough s.t. the cut has a total cost of $n\epsilon$. If $n\epsilon > 2$, it's better to cut off a single pixel of cost 2. The most influencial way to remove the bias is Normalized cut.

Grab cut: Image witha n object in the middle. Min-cut will just cut off a tiny piece. Construct a graph using something like (30). Then the user traces this is forground and and this is background. So we reconstruct the graph with terminal nodes *foreground* and *background.* If a pixel has foreground, it's weight to the foreground pixel is ∞. Since this is interactive, user can fix it up. This is more like a MRF, where edges are pairwise costs and "grabCut" does the user interaction by squares. State of the art interactive segmentation result.

Normalized-Cut A generic graph algorithm. $V = \{\text{all vertices}\}$, a *cut* divides V into A, B s.t. $A \cap B = \emptyset, A \cup B = V$. A cost of a cut is

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \tag{31}$$

Sum over u and v that are neighbors. An *association* between A and B is the total sum of weights coming out of all of A. $assoc(A, V) = \sum_{u \in A, v \in V} w(u, v)$

Proposed min cut

$$\min_{A,B} Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(B, A)}{assoc(B, V)} \tag{32}$$

For small cut, the ratio would be 1, which would be high, so we avoid cutting smaller corners.

Suppose we had an image that was all white. The best Ncut is a straight line in the middle. Vertical lines are equally low in cost so in terms of mincut that's good. By making them in the middle, you make the two terms equal. By symmetry the minimum is when those two are the same.

So this has a bias towards equal sized shapes and a more compact, convex shapes. A circle vs very snake line polygon, there'll be more cuts between A and B in the snake, so circle is better. In general it thinks circular is better. (Some disadvantages to this although the biases here is better than biases in min-cut. Still it's generally a bad thing to have a bias that you can't control becuase it won't fit the problem.)

Computation Computing this minimization is NP-hard (if you make your solution to have discrete yes foreward or no backward). You can relax the problem's discrete constraints (1 or -1), then the problem has a simple solution with eigenvalues.

Let W be the matrix of edge weights, symmetric because the graph is undirected. $W_{j,i}$ = edge between i and j. x is a solution vector and $x_i = \{1, -1\}$. d is a vector of total weights, $d_i = \sum_{j \in V} W_{ij}$ is the sum of all the edges leaving node i. D is a diagonal matrix d_i's on the diagonal. d_i is just the sum of i-th column of W.

With a lot of arithmetic manipulation, they show that $Ncut$ is equivalent to solving the following:

$$Ncut \equiv \frac{y^T(D - W)y}{y^T Dy} \tag{33}$$

where $y = (\mathbf{1} + x) - b(\mathbf{1} - x)$ and $b \in \mathbf{R}$ that depends on the segmentation, the ratio of all the edges in A leaving A and that of B. $b = \frac{assoc(A,V)}{assoc(B,V)}$. If $x_i = 1, y_i = 2$, $x_i = -1, y_i = -2b$.

Example for intuition: Suppose $b = 1$, we can divide images in to two equal sets. Suppose $\boldsymbol{x}$ has bunch of 1s followed by bunch of -1s. $x = (1, \dots, 1, -1 \dots, -1)^T, y = (2, \dots, 2, -2, \dots, -2)^T$. So $Dy = Wy = (2d_1, 2d_2, \dots, 2d_k, -2d_{k+1}, \dots, -2d_n)^t$. When we multiply W by y, we get positive values for first part, negative values for the second pard... NEXT LECTURE.

How to solve (33)**?** We let $Z = D^{1/2}y$ to remove D from the denominator. So we get $y = D^{-1/2}Z$ and now (33) becomes

$$\frac{z^t D^{-1/2}(D-W)D^{-1/2}z}{z^T z} = \frac{z^T M z}{z^T z}$$

, where $M = D^{-1/2}(D-W)D^{-1/2}$. This is still symmetric becaue both row and col are scaled in the same way. Since M is a symmetric matrix, we can decompose it into: $C = Q^T M Q$, where Q is an orthonormal matrix and C is diagonal, which is equivalent to saying $QCQ^T = M$. Let $Qv = Z$, where $v = Q^T Z$, substitue all, then we get

$$\begin{aligned} \frac{z^T M z}{z^T z} &= \frac{v^t Q^t C Q^T Q v}{v^T Q^T Q v} \\ &= \frac{vCv}{v^T v} \\ &= \frac{v_1^2 \lambda_1^2 + \cdots + v_n^2 \lambda_n^2}{v_1^2 + \cdots + v_n^2} \end{aligned}$$

Where the diagonal elements in C $\lambda_1, \lambda_2, \dots, \lambda_n$, because they are the eigenvalues of M.

This is basically a weighted average by λ_is. This is minimized by giving the smallest one the most weight. So this is minimized by $v = (0, \dots, 0, 1)$, where z is the eigenvector associated with the smallest eigenvalue of M.

Extra condition, we actually don't want the smallest eigenvalue of M because if we have a trivial segmentation where all the values in $\boldsymbol{x}$ is 1, the smallest eigenvalue would be 0 ($(D-W)y = 0$). So we want the second smallest eigenvalue. z is the eigenvector associated with the second smallest eigenvalue.

This entire process gives us a continuous values of $\boldsymbol{x}$. We pick some threshold and say that everything above is A and everything below is B. Since we really want to minimize the cost, we can try every single threshold and test which one gives us the minimum Ncut. Because there are only N possible threshold. This isn't so expensive.

Size of M is n^2 where n is the # of pixels. This is pretty expensive if n is big. To get around this, we don't have to connect every pixel to every pixel, say within 10 pixels. So every pixel is connected to 100 pixels. It will give us a million by million graph that's very sparse. But there's a method to find the eigenvalues with sparse graph.

What do they actually use for edge weights? They use $d(i,j) = \begin{cases} 1 & \text{immediate neighbor} \\ 2 & else \end{cases}$, and

$$e^{\frac{-d(i,j)^2}{\sigma_d}} e^{\frac{-(I_i, I_j)^2}{\sigma_I}}$$

This looks exactly like bilateral filter

13 Lecture - 13: Parametric Clustering

Midterm assigned 3/26, due 4/2.

13.1 Project 1

Taking the first derivative a sobel mask $(-.1, 0, .1)$. For NL-means, h is the big parameter that determines how much smoothing should be done.

13.2 Normalized Cut Example

Recap: W , edge weights between all pairs of pixels, D, a diagonal matrix where D_{ii} = sum of all weights leaving/connected to vertex i. $\boldsymbol{x}$ a vector with domain $\{-1, 1\}$. What you want to do is to

$$\min \frac{assoc(A,B)}{assoc(A,V)} + \frac{assoc(B,A)}{assoc(B,V)}$$

and this is equivalent to $\min \frac{y^T(D-W)y}{y^TDy}$, where $y = (1+\boldsymbol{x}) - b(1-\boldsymbol{x})$, $b = \frac{assoc(A,V)}{assoc(B,V)}$. Suppose $b = 1$. Then $y \in \{-2, 2\}$, everything is symmetric.

Intuition why $\min \frac{assoc(A,B)}{assoc(A,V)} + \frac{assoc(B,A)}{assoc(B,V)} \equiv \min \frac{y^T(D-W)y}{y^TDy}$: Wy, a column vector, the inner product of the first row of W (all edges leaving A) and $\boldsymbol{y}$, so $Wy = \begin{pmatrix} 2assoc(v_1, A) - 2assoc(v_1, B) \\ 2assoc(v_2, A) - 2assoc(v_2, B) \\ \vdots \end{pmatrix}$

Now, y^TWy, a quadratic equation, every edge between A and A are multiplyed by 2, and every edge between A and B are mulitplied by 2 and -2, and every edge between B and B are multiplied by 2. think about parts of that's 2, i.e. vertices that correspond to A. Each element in Wy is multiplyed by 2, you'll get 4 times $assoc(A, A)$, when -2 is multiplied by

$$y^TWy = 4assoc(A,A) - 4assoc(A,B) - 4assoc(B,A) + 4assoc(B,B).$$

First row of D gets multiplied by y,

$$\begin{aligned} y^TDy &= y^T \begin{pmatrix} y_1 assoc(v_1, V) \\ y_2 assoc(v_2, V) \\ \vdots \end{pmatrix} \\ &= 4assoc(A,V) - 4assoc(B,V) \end{aligned}$$

All of the edge weights multiplied by 4. They all cancel out so remove them. Now the ratio

$$\begin{aligned} y^TWy/y^TDy &= \frac{assoc(A,V) + assoc(B,V) - assoc(A,A) - assoc(B,B) - 2assoc(A,B)}{assoc(A,V) + assoc(B,V)} \\ &= \frac{assoc(A,B) + assoc(B,A) + assoc(A,B) + assoc(B,A)}{assoc(A,V) + assoc(B,V)} \\ &= \frac{2assoc(A,B)}{2assoc(A,V)} + \frac{2assoc(B,A)}{2assoc(B,V)} \\ &= \frac{assoc(A,B)}{assoc(A,V)} + \frac{assoc(B,A)}{assoc(B,V)} \end{aligned}$$

Which is what we wanted. Since $assoc(A, V)$ is edges coming out from all of A, if you subtract it with $assoc(A, A)$, we get $assoc(A, B)$.

Normalized cut can be applied to any graph, it's just a way of taking a graph and seperating it in two parts. If two nodes (pixels) are similar, we have a stronger edge between them.

Going more abstract, think about the problem as clustering points in high dimensional space: K-means, EM

13.3 Parametric Clustering

Few examples: a bunch of points in a 3-D space, and we want to group these points together in a cluster. A good cluster is a cluster where the points are close togehter as possible given a limited number of clusters. In vision, your three dimensions might be RGB. You might want to cluster according to colors. If you discretize those clusters, finding segments might be easier. Another reason you might want to do this is to add 2 more dimensions, RBG + X and Y. Then clusters are those similar in color and space. This can be very effective. Another example is a perceptual grouping problem. If you draw three lines (2 with positive slope, 1 intersecting both), there are 3 clusters and the points are close to those lines. You can represent things parametrically. For a line, you need n-1 to represent direction, n-1 as the shift $= 2(n-1)$ parameters. Looking at lines can give you structures of a building, another reason for looking at lines is to estimate the plane. If you move a camera in one direction and if you track points, in the image, objects move in the opposite direction but background moves slower. As camera moves, points on a plane moves in affine transformations, so you can cluster points in similar affine transformation (6 free degrees). These are all parametric clustering problems.

Technically this is known as the chicken and egg problems. If we knew the parameters, we can figure out the assignments to the cluster, if you know the assignments you can figure out the parameters that fit. The way to go around this is iterative method, where you have an initial guess and you keep updating until you converge. Many of these problems might be NP-hard to find the optimal solution, so we use heuristics which is the iterative algorithm. Although this is heuristics in practice it often works very well.

K-means We have bunch of points $x_1, \ldots, x_n$, and we want to form K clusters $A_1, \ldots, A_k$, where $x_i \in A_j$ means point i is in cluster j. Ever cluster will have a parameter that describes it and here we'll use cluster centers, $c_1, \ldots, c_k$. Our objective is

$$\min_{A,c} \sum_{j=1}^{k} \sum_{x_i \in A_j} ||c_j - x_i||^2 \tag{34}$$

The algorithm:

1. Guess centers c
2. Assign each point to nearest center
3. Recompute c s.t. $c_j = \sum_{x_i \in A_j} \frac{x_i}{||A_j||}$
4. Repeat until convergence. (no assignments change)

The reassignment can only reduce the cost. If you want to minimize $\sum_{x_i \in A_j} ||c_j - x_i||^2$, you can take the derivatie but you find out that the center is the average of all the points in the cluster. Convergence? It might not converge to a global minima but it will converge to a local minima. We'll have a cycle if there's a point half way in between and it keeps on changing. But cycling doesn't change the cost, so if convergence is defined by when cost doesn't change, convergence is assured.

Used in color quantization, when you want to compactly represent a colro image. Every pixel has RBG, which is 24 bits. Suppose you want to represent 4 bits not 24, this means we only have 16 colors. You can quantize it and it'll still look good. For something like this we don't necessarily have to have the global clustering.

The first guess in K-means is really important. You can get situations where all of your points gets assinged to a single cluster. You need good heuristics to chose the starting guess.

14 Lecture - 14: EM

Midterm Due 4/4/12. You can talk about the material, using the internet as a resource is fine, anything you find. Cite any material used outside the class website Questions

1. G_σ a gaussian filter with std σ, unknown. Use it to filter a cosine function. Figure out what's the value of filtered functions at other points.

2. Apply perona-malik to a sine wave, what is the method noise? Idea is if the filter is perfect the signal won't change. But it does, have a sense of how much method changes the original signal

3. Prove the method noise of non-local means is white noise. Apply NL-means to sine wave (the analytic expression of the method noise is a pain), look at the method noise of two different locations, show that they are uncorrelated even if those points are close. Prove true or false. **Extra credit** if you can characterize the method noise intuitively.

4. MRF, a rabbit moving from one region to a neighboring region with equal probability, (never stay at the same region). After a while what's the probability distribution (stable distriubtion?) Give a numerical distribution (numbers).

5. MRF 4 labels for every pixel. Create a gibbs distribution using the provided priors (Next to each other means the 4 connected neighbors). Then optimize this by a graph cut algorithm, can we get into a local optimum that's not global? give a proof that this won't happen, or an example where it will get stuck in a local optima.

6. Given an N by N image with a grid structured graph (4 connected), with weight 1 for all edges. Suppose there is an rectangle that doesn't touch the image boundary (A by B)

   (a) what's the Ncut cost of doing this two-part seperation.

   (b) (Not saying normalized cut, he's asking us about the global solution, not the solution Ncut produces) Think about what gives you the global solution. Intuition is if you applied Ncut that finds the global minimum, does it find the rectangle? If not, can we change the cost so that this rectangle is the global solution?

   (c) Has nothing to do with (b). Prove that rectangle that optimize this cost globally is actually a square. A and B have to be integers, so you can formulate the graph in continuous case and show that it's a square in continuous case. Let R the A by B rectangle, and the rest $I - R$, if you find $ncut(R, I - R)$ (global minimum) is R a square?

14.1 EM

EM is just like K-means but rather than doing hard assignments does soft assignments. Instead of taking a point and assigning it to a cluster, you assign partial values of how sure the point belongs to all clusters. When recomputing the center, take the weighted average of confidence values. This helps us not converge into a bad local minima.

Another way of looking at EM from Bayesian pov. We are going to assume that the points were generated by a mixture of gaussians. The points were generated by one of these gaussians. Fit a mixture of gaussians to our points i.e. fitting a probability distribution to data. Each cluster is assigned a different gaussian distribution. Each gaussian with different size, mean, and co-variance. The real reason why we use a mixture of gaussians is not because we think it's accurate but because of it's nice properties we just want to use the most simplest distribution that we can get away with.

Mixture of Gaussians (Assume for simplicity covariance is a real number, gaussian circular) Formally,

$$P(x; \boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{k=1}^{K} p_k g(x; \mu_k, \sigma_k)$$

We have K gaussians each with its own μ and σ. Now, given some data we want to fit some distribution to it. Let $\boldsymbol{\pi}$ be the vector of all p_k values. What we want to do is

$$\max_{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma}} \sum_n P(x_n; \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi})$$

Maximum likelyhood approximation. This would be easy if we knew which gaussian a point belongs to. Let $z_{k,n} = 1$ iff $x_n \in G_k$ and 0 otherwise. So z is a hidden varialbe that tells you which distribution (gaussian) this point comes from. Now we can re-write the problem as

$$\max_{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma}} \sum_n \sum_k P(x_n; \mu_k, \sigma_k, \pi_k) z_{k,n} \tag{35}$$

Now, replace $z_{k,n}$ with $\mathbf{E}(z_{k,n}; \mu_k, \sigma_k, \pi_k)$. Now you iterate between fixing paramters, finding $\mathbf{E}z_{k,n}$, then using that maximizing the equation by turing the parameters.

So,

E step:

$$z^i_{k,n} = \frac{p^i_k g(x_n; \mu^i_k, \sigma^i_k)}{\sum_k p^i_k g(x_n; \mu^i_k, \sigma^i_k)} \tag{36}$$

Probability that distribution k generated (p_k is the probability that the point came from the k-th gaussian) normailzed is the expectation.

M step:

$$\mu^{i+1}_k = \frac{\sum_n z^i_{k,n} x_n}{\sum_n z_{k,n}} \tag{37}$$

Just the average, weighted by the expectation of where n-th point comes from. Just like the sample mean. Exactly the same thing for vaiance.

$$p^{i+1}_k - \frac{1}{N} \sum z_{k,n} \tag{38}$$

In stead of taking the expectation of z if we took the maximum this would be k-means.

Kernel Density Estimation Suppose we have a bunch of points and we want to build a probability distribution that fits the data. Instead of fitting it in mixture of gaussians, put a small guassian distribution on every sample, so if there are n points, we get a mixture of n gaussians. The advantage of thsi is that in the limit as $n \to \infty$ and make gaussians smaller and smaller, the limit approaches the true distribution (trusting the data). The disadvantage is that this requries a lot of parameters, each point is a parameter.

A combination is **mean-shift** segmentation algorithm. Take your data, model it as kernel density estimation, then find the modes of this distribution and treat those as cluster centers. Modes are the points of local maximum (the biggest value, the peaks). Intuition is starting off at one point, then imagine that point is a center of a gaussian distribution, weight all of the points, and take the weighted mean of all points (just like the maximization step in EM), then move to the new mean and keep doing this. If you keep on doing this this converges to the mode of a kernel density estimation. It's like EM with only one class.