Fitting: Voting and the Hough Transform

Monday, Feb 14
Prof. Kristen Grauman
UT-Austin

Last time: Grouping

- Bottom-up segmentation via clustering
  - To find mid-level regions, tokens
  - General choices -- features, affinity functions, and clustering algorithms
- Example clustering algorithms
  - Mean shift and mode finding: K-means, Mean shift
  - Graph theoretic: Graph cut, normalized cuts
- Grouping also useful for quantization
  - Texton histograms for texture within local region

Recall: Images as graphs

- Fully-connected graph
  - node for every pixel
  - link between every pair of pixels, p,q
  - similarity $w_{pq}$ for each link
  - similarity is inversely proportional to difference in color and position

Goal: Segmentation by Graph Cuts

- Break graph into segments
  - Delete links that cross between segments
  - Easiest to break links that have low similarity
    - similar pixels should be in the same segments
    - dissimilar pixels should be in different segments

Last time: Measuring affinity

- Points $x_1, x_{10}$
- $40 \times 40$ affinity matrix $A$

$$A(i,j) = \exp(-\frac{1}{2\sigma^2}||x_i - x_j||^2)$$

1. What do the blocks signify?
2. What does the symmetry of the matrix signify?
3. How would the matrix change with larger value of $\sigma$?

Example: weighted graphs

- Suppose we have a 4-pixel image (i.e., a $2 \times 2$ matrix)
- Each pixel described by 2 features

Dimension of data points : $d = 2$
Number of data points : $N = 4$
for $i=1:N$  
for $j=1:N$  
$D(i,j) = ||x_i - x_j||^2$  
end  
end

Example: weighted graphs
Computing the distance matrix:

$D(1,:) = (0.24 \ 0.01 \ 0.47)$

Example: weighted graphs
Computing the distance matrix:

$D(1,:) = (0.24 \ 0.01 \ 0.47)$

Example: weighted graphs
Computing the distance matrix:

$N \times N$ matrix

Example: weighted graphs
Computing the distance matrix:

$D \rightarrow \text{affinities}$

Scale parameter $\sigma$ affects affinity

Distance matrix

Affinity matrix with increasing $\sigma$:

Visualizing a shuffled affinity matrix
If we permute the order of the vertices as they are referred to in the affinity matrix, we see different patterns:
Putting these two aspects together

\[ A(i,j) = \exp\left(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2\right) \]

Cuts in a graph: Min cut

Link Cut
- set of links whose removal makes a graph disconnected
- cost of a cut:
  \[ \text{cut}(A,B) = \sum_{p \in A, q \in S} w_{pq} \]

Find minimum cut
- gives you a segmentation
- fast algorithms exist

Cuts in a graph: Normalized cut

- Fix bias of Min Cut by normalizing for size of segments:
  \[ N\text{cut}(A,B) = \frac{\text{cut}(A,B)}{\text{assoc}(A,V)} \times \frac{\text{cut}(A,B)}{\text{assoc}(B,V)} \]
  \[ \text{assoc}(A,V) = \text{sum of weights of all edges that touch } A \]

- Ncut value is small when we get two clusters with many edges with high weights, and few edges of low weight between them.

Example results: segments from Ncuts

Normalized cuts: pros and cons

Pros:
- Generic framework, flexible to choice of function that computes weights ("affinities") between nodes
- Does not require model of the data distribution

Cons:
- Time complexity can be high
  - Dense, highly connected graphs \( \rightarrow \) many affinity computations
  - Solving eigenvalue problem
- Preference for balanced partitions

Segments as primitives for recognition

B. Russell et al., "Using Multiple Segmentations to Discover Objects and their Extent in Image Collections," CVPR 2006

Note: Citations and references are not fully captured in this transcription.
Motion segmentation


Now: Fitting

• Want to associate a model with observed features

For example, the model could be a line, a circle, or an arbitrary shape.

Fitting: Main idea

• Choose a parametric model to represent a set of features
• Membership criterion is not local
  • Can’t tell whether a point belongs to a given model just by looking at that point
• Three main questions:
  • What model represents this set of features best?
  • Which of several model instances gets which feature?
  • How many model instances are there?
• Computational complexity is important
  • It is infeasible to examine every possible set of parameters and every possible combination of features

Example: Line fitting

• Why fit lines?
  Many objects characterized by presence of straight lines

• Wait, why aren’t we done just by running edge detection?

Difficulty of line fitting

• Extra edge points (clutter), multiple models:
  • which points go with which line, if any?
• Only some parts of each line detected, and some parts are missing:
  • how to find a line that bridges missing evidence?
• Noise in measured edge points, orientations:
  • how to detect true underlying parameters?

Voting

• It’s not feasible to check all combinations of features by fitting a model to each possible subset.
• Voting is a general technique where we let the features vote for all models that are compatible with it.
  • Cycle through features, cast votes for model parameters.
  • Look for model parameters that receive a lot of votes.
• Noise & clutter features will cast votes too, but typically their votes should be inconsistent with the majority of “good” features.
Fitting lines: Hough transform

- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?

Hough Transform is a voting technique that can be used to answer all of these questions.

Main idea:
1. Record vote for each possible line on which each edge point lies.
2. Look for lines that get many votes.

Finding lines in an image: Hough space

Connection between image (x,y) and Hough (m,b) spaces
- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that \( y = mx + b \)

Finding lines in an image: Hough algorithm

How can we use this to find the most likely parameters (m,b) for the most prominent line in the image space?
- Let each edge point in image space vote for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.
Hough line demo

http://www.dis.uniroma1.it/~iocchi/slides/icra2001/javahough.html

Hough transform algorithm

Using the polar parameterization:
\[ x \cos \theta - y \sin \theta = d \]

Basic Hough transform algorithm
1. Initialize \( H[d, \theta] = 0 \)
2. for each edge point \((x, y)\) in the image
   for \( \theta = \theta_{\text{min}} \) to \( \theta_{\text{max}} \) // some quantization
   \[ d = x \cos \theta - y \sin \theta \]
   \[ H[d, \theta] += 1 \]
3. Find the value(s) of \((d, \theta)\) where \( H[d, \theta] \) is maximum
4. The detected line in the image is given by \( d = x \cos \theta - y \sin \theta \)

Time complexity (in terms of number of votes per pt)?

\[ dyx = \frac{1}{\sin \theta \cos \theta} \]

Source: Steve Seitz

Impact of noise on Hough

Here, everything appears to be "noise," or random edge points, but we still see peaks in the vote space.
Extensions

Extension 1: Use the image gradient
1. same
2. for each edge point \([x,y]\) in the image
   \[ \theta = \text{gradient at } (x,y) \]
   \[ d = x \cos \theta - y \sin \theta \]
   \[ H[d, \theta] += 1 \]
3. same
4. same
(Reduces degrees of freedom)

Extension 2
• give more votes for stronger edges (use magnitude of gradient)

Extension 3
• change the sampling of \((d, \theta)\) to give more/less resolution

Extension 4
• The same procedure can be used with circles, squares, or any other shape...

---

Hough transform for circles

• Circle: center \((a,b)\) and radius \(r\)
  \[(x - a)^2 + (y - b)^2 = r^2\]

• For a fixed radius \(r\), unknown gradient direction

Intersection:
most votes for center occur here.

---

Hough transform for circles

• Circle: center \((a,b)\) and radius \(r\)
  \[(x - a)^2 + (y - b)^2 = r^2\]

• For an unknown radius \(r\), unknown gradient direction

Intersection: most votes for center occur here.
Hough transform for circles

- Circle: center \((a, b)\) and radius \(r\)
  \[(x - a)^2 + (y - b)^2 = r^2\]
- For an unknown radius \(r\), known gradient direction

\[
\begin{align*}
H[0, 0, r] &= 1 \\
& \quad \text{end}
\end{align*}
\]

For every edge pixel \((x, y)\):
For each possible radius value \(r\):
For each possible gradient direction \(\theta\):
// or use estimated gradient at \((x, y)\)
\[
\begin{align*}
a &= x - r \cos(\theta) \quad \text{// column} \\
b &= y + r \sin(\theta) \quad \text{// row}
\end{align*}
\]
\[H[a, b, r] += 1\]
\[\text{end}\]
\[\text{end}\]
\[\text{end}\]

Check out online demo: http://www.markschulze.net/java/hough

Time complexity per edgel?

Kristen Grauman

Example: detecting circles with Hough

Original Edges Votes: Penny

Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Example: iris detection

Hemerson Pistori and Eduardo Rocha Costa


Kristen Grauman
Voting: practical tips

• Minimize irrelevant tokens first
• Choose a good grid / discretization
  
  Too fine  ?  Too coarse
  
• Vote for neighbors, also (smoothing in accumulator array)
• Use direction of edge to reduce parameters by 1
• To read back which points voted for “winning” peaks, keep tags on the votes.

Hough transform: pros and cons

Pros
• All points are processed independently, so can cope with occlusion, gaps
• Some robustness to noise: noise points unlikely to contribute consistently to any single bin
• Can detect multiple instances of a model in a single pass

Cons
• Complexity of search time increases exponentially with the number of model parameters
• Non-target shapes can produce spurious peaks in parameter space
• Quantization: can be tricky to pick a good grid size

Generalized Hough Transform

• What if we want to detect arbitrary shapes?

Intuition:

\[ \text{Displacement vectors} \]
\[ \text{Ref. point} \]
\[ \text{Model image} \]
\[ \text{Novel image} \]
\[ \text{Vote space} \]

Now suppose those colors encode gradient directions...

Generalized Hough for object detection

• Instead of indexing displacements by gradient orientation, index by matched local patterns.

\[ \text{training image} \]
\[ \text{“visual codeword” with displacement vectors} \]

B. Leibe, A. Leonardis, and B. Schiele, Combined Object Categorization and Segmentation with an Implicit Shape Model, ECCV Workshop on Statistical Learning in Computer Vision 2004

Source: L. Lazebnik
Generalized Hough for object detection

- Instead of indexing displacements by gradient orientation, index by “visual codeword”

**Summary**

- **Grouping/segmentation** useful to make a compact representation and merge similar features
  - associate features based on defined similarity measure and clustering objective

- **Fitting** problems require finding any supporting evidence for a model, even within clutter and missing features.
  - associate features with an explicit model

- **Voting** approaches, such as the Hough transform, make it possible to find likely model parameters without searching all combinations of features.
  - Hough transform approach for lines, circles, …, arbitrary shapes defined by a set of boundary points, recognition from patches.

**References**

- B. Leibe, A. Leonardis, and B. Schiele, Combined Object Categorization and Segmentation with an Implicit Shape Model, ECCV Workshop on Statistical Learning in Computer Vision 2004
- Source: L. Lazebnik

**Test Image**

Kristen Grauman