Visual Interpretation using Probabilistic Grammars

Paul Robertson
Model-Based Vision

- What do the models look like
- Where do the models come from
- How are the models utilized
The Problem
Optimization/Search Problem

Find the most likely interpretation of the image contents that:

1. Identifies the component parts of the image correctly.
2. Identifies the scene type.
3. Identifies structural relationships between the parts of the image.

Involves: Segmenting into parts, naming the parts, and relating the parts.
Outline

• Overview of statistical methods used in speech recognition and NLP
• Image Segmentation and Interpretation
  – image grammars
  – image grammar learning
  – algorithms for parsing patchwork images.
Not any description – the best

Bad parse

Good parse
What’s similar/different between image analysis and speech recognition/NLP?

- **Similar**
  - An input signal must be processed.
  - Segmentation.
  - Identification of components.
  - Structural understanding.
- **Dissimilar**
  - Text is a valid intermediate goal that separates Speech recognition and NLP. Line drawings are less obviously useful.
  - Structure in images has much more richness.
Speech Recognition and NLP

- Little backward flow
- Stages done separately.
- Similar techniques work well in each of these phases.
- A parallel view can also be applied to image analysis.
Speech Understanding

• Goal: Translate the input signal into a sequence of words.
  – Segment the signal into a sequence of samples.
    • $A = a_1, a_2, ..., a_m \quad a_i \in \mathcal{A}$
  – Find the best words that correspond to the samples based on:
    • An acoustic model.
      – Signal Processing
        – Prototype storage and comparator (identification)
    • A language model.
      • $W = w_1, w_2, ..., w_m \quad w_i \in \mathcal{V}$
  – $W_{opt} = \arg \max_w P(W|A)$
  – $W_{opt} = \arg \max_w P(A|W) P(W)$
    • (since $P(W|A) = P(A|W) P(W) / P(A)$ [Bayes])
    • $P(A|W)$ is the acoustic model.
    • $P(W)$ is the language model.
language modeling for speech

\[
P(W) = \prod_{i=1}^{n} P(w_i \mid w_1, \ldots, w_{i-1})
\]

\[
P(W) = \prod_{i=1}^{n} P(w_i \mid \Phi(w_1, \ldots, w_{i-1}))
\]

\[
P(W) = \prod_{i=1}^{n} P(w_i \mid \Phi_{i-1})
\]

\[
P(w_i \mid w_{i-1}, w_{i-2}) = f(w_i \mid w_{i-1}, w_{i-2})
\]

\[
P(w_i \mid w_{i-1}, w_{i-2}) = \lambda_3 f(w_i \mid w_{i-1}, w_{i-2}) + \lambda_2 f(w_i \mid w_{i-1}) + \lambda_1 f(w_i)
\]

\[
\lambda_1 + \lambda_2 + \lambda_3 = 1
\]

- Using the above
  - \( P(W) \) can be represented as a HMM and solved efficiently using the Viterbi algorithm.
  - The good weights \( \lambda_1, \lambda_2, \) and \( \lambda_3 \) can be computed using the Baum-Welch algorithm.
Natural Language Processing

• Part of correctly *understanding* a sentence comes from correctly *parsing* it.

• Starting with a word list, parsing involves two separable activities:
  
  – Part of speech tagging.
    • Find the most *probable* assignments of parts of speech.
  
  – Parsing the words into a tree.
    • Find the most *probable* parse tree.
Part-of-speech tagging

• Goal: Assign part-of-speech tags to each word in the word sequence.
  – Start with the word sequence
    • $W = w_1, w_2, ..., w_m$ \( w_i \in \mathcal{Y} \)
  – Find the best tags for each word
    • $T = t_1, t_2, ..., t_m$ \( t_i \in \mathcal{I} \)
\[ P(w_1, n) = \sum_{t_1, n+1} P(w_1, n, t_1, n + 1) \]

\[ T_{opt} = \arg \max_{t_1, n} P(t_1, n \mid w_1, n) \]

\[ T_{opt} = \arg \max_{t_1, n} P(t_1, n, w_1, n) \]

\[ P(w_n \mid w_1, n-1, t_1, n) = P(w_n \mid t_n) \]

\[ P(t_n \mid w_1, n-1, t_1, n-1) = P(t_n \mid t_{n-1}) \]

\[ P(w_1, n) = \sum_{t_1, n+1} \prod_{i=1}^{n} P(w_i \mid t_i)P(t_{i+1} \mid t_i) \]

\[ P(w_1, n) = \sum_{t_1, n+1} \prod_{i=1}^{n} P(w_i \mid t_i)P(t_{i+1} \mid t_i, t_{i-1}) \]

- \( T_{opt} \) is the path the HMM traverses in producing the output (since the states of the HMM are the tags).
- Use Viterbi algorithm to find the path.
PCFG’s

• Better language models lead to better results.
• Considering the grammar instead of a simple sequence of words, the relationships are more meaningful.
• PCFG is \( <W, N, N^1, R> \)
  – \( W \) is a set of terminal symbols
  – \( N \) is a set of non-terminal symbols
  – \( N^1 \) is the starting symbol
  – \( R \) is a set of rules.
    • Each rule \( N^i \rightarrow \text{RHS} \) has an associated probability \( P(N^i \rightarrow \text{RHS}) \)
      which is the probability of using this rule to expand \( N^i \)
• The probability of a sentence is the sum of the probabilities of all parses.
• Probability of a parse is the product of the probabilities of all the productions used.
• Smoothing necessary for missing rules.
Example PCFG

\[
\begin{align*}
s & \rightarrow \text{ np } \text{ vp } & 0.8 \\
s & \rightarrow \text{ vp } & 0.2 \\
\text{np} & \rightarrow \text{ noun } & 0.4 \\
\text{np} & \rightarrow \text{ noun } \text{ pp } & 0.4 \\
\text{np} & \rightarrow \text{ noun } \text{ np } & 0.2 \\
\text{vp} & \rightarrow \text{ np } \text{ vp } & 0.3 \\
\text{vp} & \rightarrow \text{ np } \text{ vp } & 0.3 \\
\text{vp} & \rightarrow \text{ np } \text{ vp } & 0.2 \\
\text{vp} & \rightarrow \text{ np } \text{ vp } & 0.2 \\
\text{pp} & \rightarrow \text{ prep } \text{ np } & 1.0 \\
\text{prep} & \rightarrow \text{ like } & 1.0 \\
\text{verb} & \rightarrow \text{ swat } & 0.2 \\
\text{verb} & \rightarrow \text{ flies } & 0.4 \\
\text{verb} & \rightarrow \text{ like } & 0.4 \\
\text{noun} & \rightarrow \text{ swat } & 0.1 \\
\text{noun} & \rightarrow \text{ flies } & 0.4 \\
\text{noun} & \rightarrow \text{ ants } & 0.5
\end{align*}
\]

- Good parse = $0.2 \times 0.2 \times 0.2 \times 0.4 \times 0.4 \times 1.0 \times 1.0 \times 0.4 \times 0.5 = 0.000256$
- Bad parse = $0.8 \times 0.2 \times 0.4 \times 0.1 \times 0.4 \times 0.3 \times 0.4 \times 0.4 \times 0.5 = 0.00006144$
Why these techniques are dominating language research

- **Statistical methods work well**
  - The best POS taggers perform close to 97% accuracy compared to human accuracy of 98%.
  - The best statistical parsers are at around 88% vs an estimated 95% for humans.

- **Learning from the corpus**
  - The grammar can be learned from a representative corpus.

- **Basis for comparison**
  - The availability of corpora with ground truth enables researchers to compare their performance against other published algorithms/models.

- **Performance**
  - Most algorithms at runtime are fast.
Build Image Descriptions
Patchwork Parsing

• Use semantic segmentation to produce a set of homogeneous regions
• Based on the contents of the regions and their shape hypothesize region contents.
• Region contents is ambiguous in isolation
  – Use contextual information to reduce ambiguity.
• The image must make sense
  – We must be able to produce a parse for it.
• Our interpretation of the image approximates the most probable parse.
  – Success of the picture language model determines whether most-probable-parse works.
• Do it (nearly) as well as human experts
Segmented image labeling

- The image contains $n$ regions $r_{1,n}$.
- Each region has a set of neighbors $n_{1,n}$.
- $P(r_{1,n})$ is the sum of the disjoint labelings.

\[ P(r_{1,n}) = \sum_{l_{1,n}} P(r_{1,n}, l_{1,n}) \]
• We wish to find the labeling $L_{1,n}$.

$$L_{1,n} = \arg \max_{l_{1,n}} \prod_{i=1}^{n} P(l_i \mid r_i, n_i)$$

$$= \arg \max_{l_{1,n}} \prod_{i=1}^{n} \frac{P(l_i \mid r_i)P(n_i \mid l_i, r_i)}{P(n_i \mid r_i)}$$

$$= \arg \max_{l_{1,n}} \prod_{i=1}^{n} \frac{P(l_i \mid r_i)P(n_i \mid l_i)}{P(n_i \mid r_i)}$$

$$= \arg \max_{l_{1,n}} \prod_{i=1}^{n} P(l_i \mid r_i)P(n_i \mid l_i)$$

• $P(l_i \mid r_i)$ is the optical model.
• $P(n_i \mid l_i)$ is the picture language model.
Segmentation
The optical model

- Filters produce useful features from the original image.
- Semantic Segmentation produces regions.
- Prototype database and comparator produce evidence for labeling each region.

```
(setq *region-optical-evidence*
  '((r1 (field . 0.5) (swamp . 0.2) (town . 0.1) (lake . 0.1) (road . 0.05) (river . 0.05))
    (r2 (field . 0.5) (swamp . 0.2) (town . 0.1) (lake . 0.1) (road . 0.05) (river . 0.05))
    (r3 (field . 0.5) (swamp . 0.2) (town . 0.1) (lake . 0.1) (road . 0.05) (river . 0.05))
    (r4 (field . 0.1) (swamp . 0.1) (town . 0.1) (lake . 0.3) (road . 0.3) (river . 0.1))
    (r5 (field . 0.1) (swamp . 0.1) (town . 0.3) (lake . 0.1) (road . 0.3) (river . 0.1))
    (r6 (field . 0.1) (swamp . 0.1) (town . 0.1) (lake . 0.3) (road . 0.1) (river . 0.3))
    (r7 (field . 0.3) (swamp . 0.4) (town . 0.1) (lake . 0.1) (road . 0.05) (river . 0.05))
    (r8 (field . 0.3) (swamp . 0.4) (town . 0.1) (lake . 0.1) (road . 0.05) (river . 0.05))
    (r9 (field . 0.1) (swamp . 0.2) (town . 0.5) (lake . 0.1) (road . 0.05) (river . 0.05))
))
```

\[
R = \{< r_1, \{< l_1, P(l_1 \mid r_1) >, \ldots \} >, \ldots \}
\]

\[
\forall r_i \in R : \sum_{j=1}^{n} P(l_j \mid r_i) \leq 1
\]
• Regions have internal and external neighbors.

• Rule for a region looks this:

\[
\langle \text{Label, Internal, External, Probability} \rangle
\]

\[
\langle \text{Field, (I}_1, I_2, \ldots I_n), (E_1, E_2, E_3, E_4, \ldots E_n), 0.3 \rangle
\]
- Regions may be occluded.
- Rule for a region looks this:

\[
\langle \text{Field, } (*, I_n), (*, E_2, E_3, E_4, \ldots E_n), 0.3 \rangle
\]
Structured Regions
Example rules

- $P_1$: `<lake, (), (field), 1.0>
- $P_2$: `<field, (lake, *), (road *), 0.33>
- $P_3$: `<field, (*), (*, road, town, river), 0.33>
- $P_4$: `<field, (*), (*, river, swamp), 0.33>
- $P_5$: `<swamp, (*), (* field river), 0.5>
- $P_6$: `<swamp, (*), (* river town road), 0.5>
- $P_7$: `<river, (*), (* field town swamp * swamp field), 1.0>
- $P_8$: `<town, (), (field road swamp river), 1.0>`
Supervised Learning
Smoothing and occlusion

- Whenever we generate a rule, we also make rules for degenerate cases.
  
  <Field, (), (E₁, E₂, E₃), p?>
  <Field, (), (*, E₂, E₃), p?>
  <Field, (), (E₁, *, E₃), p?>
  <Field, (), (E₁, E₂, *), p?>
  <Field, (), (*, E₃), p?>
  <Field, (), (*, E₂), p?>
  <Field, (), (*, E₁), p?>

- Represent grammar as a lattice of approximations to the non-occluded rule.
A successful parse:

\[ ((\text{r4 Lake} () (\text{Fields1}) p1) (\text{Fields1} (\text{Lake}) (\text{Road} *) p2) (\text{Fields3} () (\text{River Town Road} *) p3) (\text{Town} () (\text{swamp2 River Field1}) p8) (\text{River} () (\text{Fields3 Town Swamp2 Swamp1 Fields2} *) p7) (\text{Swamp2} () (\text{Town Road River} *) p6) (\text{Swamp1} () (\text{River Fields} *) p5) (\text{Fields2} () (\text{River Swamp1} *) p4)) \]

Probability of image:

\[ P(\text{Lake}|r_4)P(p_1)P(\text{Field}|r_3)P(p_2)P(\text{Field}|r_2)P(p_3)P(\text{Field}|r_1)P(p_4)P(\text{Swamp}|r_7)P(p_5)P(\text{Swamp}|r_8)P(p_6)P(\text{River}|r_6)P(p_7)P(\text{Town}|r_9)P(p_8) \]
Segmenting the rule sets
Network Search Parse

• Find parses in order or probability.
• Keep sorted list of partial parses (most probably first):
  – < bindings, unprocessed regions, probability>
• Start with:
  – (<(), (r1,r2,r3,r4,r5,r6,r7,r8,r9), 1.0>)
• At each step extend the most probable:
  – (<(r2=river, r5=swamp, r8=road, r6=field, r9=town)
    (r2,r3,r4,r5,r6,r7,r8,r9) 0.5> ...)
• When applying a rule bound regions must match, unbound regions are bound.
• First attempt to extend a parse that has a null “unprocessed regions” is the most probably parse.
Network Search Performance

At each stage if there are $m$ possible labelings of the region, and for each labeling if there are $k$ rules, then for an image with $n$ regions the cost of the network search parsing algorithm is:

- $O((k*m)^n)$

Even with only 9 regions, 9 rules, and 6 possible labelings per region there are of the order of $10^{15}$ candidates.

Algorithm only terminates on VERY small examples.
Monte-Carlo Parse

- Select a complete parse at random as follows:
  (dotimes (i N)
    (start-new-parse)
    (dolist (r region-list)
      (setq l (select-at-random (possible-labels-of r)))
      (setq r (select-at-random (rules-that-generate l)))
      (store-random-parse))
- Most frequently occurring parse will approach the most probable parse as \( N \) is increased.
- How big does \( N \) have to be?
Example Monte-Carlo Parse

>> (parse-image-mc *all-regions* *rules* *region-optical-evidence*)
(((L1 . LAKE) (F1 . FIELD) (IM . IMAGE1) (RD . RIVER)
(S2 . SWAMP) (F3 . ROAD) (TN . TOWN) (F2 . RIVER) ...) NIL 4.2075E-9)

>> (dotimes (i 100) (next-parse-mc))
NIL
>> (first (setq *monte-carlo-parses* (sort *monte-carlo-parses* by-third)))
(((L1 . LAKE) (IM . IMAGE1) (S2 . SWAMP) (F1 . FIELD)
(RD . ROAD) (TN . TOWN) (F3 . FIELD) (RV . RIVER) ...) NIL 1.5147E-6)

>> (dotimes (i 100) (next-parse-mc))
NIL
>> (first (setq *monte-carlo-parses* (sort *monte-carlo-parses* by-third)))
(((F2 . FIELD) (S2 . SWAMP) (IM . IMAGE1) (F1 . FIELD)
(L1 . LAKE) (S1 . SWAMP) (RV . RIVER) (RD . ROAD) ...) NIL 2.4257475E-6)
>> (dotimes (i 100) (next-parse-mc))
NIL

>> (first (setq *monte-carlo-parses* (sort *monte-carlo-parses* by-third)))
(((F2 . FIELD) (S2 . SWAMP) (IM . IMAGE1) (F1 . FIELD)
(L1 . LAKE) (S1 . SWAMP) (RV . RIVER) (RD . ROAD) ...) NIL 2.4257475E-6)
>>
Monte-Carlo Performance

- Iterate until standard deviation $< \varepsilon$
  - As each sample is generated compute its probability.
  - Compute the standard deviation of the sample probabilities.

- We can make the error arbitrarily small by picking arbitrarily small $\varepsilon$.

- Best parse is the one from the sample with the highest probability.
  
  ```lisp
  (while (> (standard-deviation samples) epsilon)
      (start-new-parse)
      (dolist (r region-list)
          (setq l (select-at-random (possible-labels-of r)))
          (setq r (select-at-random (rules-that-generate l))))
      (store-random-parse))
  ```
Monte-Carlo Parsing Performance

- Errors
- Probability

Trials: 1 21 41 61 81 101 121 141 161 181 201

Errors: 10 8 6 4 2 0

Probability: 2.50E-06 2.00E-06 1.50E-06 1.00E-06 5.00E-07 0.00E+00
Example of correctly parsed image