Decoding, Continued

September 5, 2013
Lecture Outline

✓ Viterbi algorithm
✓ Decoding more generally

3. Five views
   ✓ MPE/MAP inference in a graphical model
2. Polytopes
“Parts”

• Assume that feature function \( g \) breaks down into local parts.

\[
g(x, y) = \sum_{i=1}^{\#parts(x)} f(\Pi_i(x, y))
\]

• Each part has an alphabet of possible values.
  – Decoding is choosing values for all parts, with consistency constraints.
  – (In the graphical models view, a part is a clique.)
Example

- One part per word, each is in \{B, I, O\}
- No features look at multiple parts
  - Fast inference
  - Not very expressive
Example

- One part per bigram, each is in \{BB, BI, BO, IB, II, IO, OB, OO\}

- Features and constraints can look at pairs
  - Slower inference
  - A bit more expressive
Let $z_{i,\pi}$ be 1 if part $i$ takes value $\pi$ and 0 otherwise.

- $z$ is a vector in $\{0, 1\}^N$
  - $N = \text{total number of localized part values}$
  - Each $z$ is a vertex of the unit cube
Score is Linear in $z$

$$\arg\max_y w^\top g(x, y) = \arg\max_y w^\top \sum_{i=1}^{\#\text{parts}(x)} f(\Pi_i(x, y))$$

$$= \arg\max_y w^\top \sum_{i=1}^{\#\text{parts}(x)} \sum_{\pi \in \text{Values}(\Pi_i)} f(\pi) 1\{\Pi_i(x, y) = \pi\}$$

$$= \arg\max_{z \in \mathcal{Z}_x} \sum_{i=1}^{\#\text{parts}(x)} \sum_{\pi \in \text{Values}(\Pi_i)} f(\pi) z_{i, \pi}$$

$$= \arg\max_{z \in \mathcal{Z}_x} F_x z$$

$$= \arg\max_{z \in \mathcal{Z}_x} (w^\top F_x) z$$
Polyhedra

- Not all vertices of the $N$-dimensional unit cube satisfy the constraints.
  - E.g., can’t have $z_{1,B_I} = 1$ and $z_{2,B_I} = 1$
- Sometimes we can write down a small (polynomial number) of linear constraints on $z$.
- Result: linear objective, linear constraints, integer constraints ...
Integer Linear Programming

• Very easy to add new constraints and non-local features.
• Many decoding problems have been mapped to ILP (sequence labeling, parsing, ...), but it’s not always trivial.
• NP-hard in general.
  – But there are packages that often work well in practice (e.g., CPLEX)
  – Specialized algorithms in some cases
  – LP relaxation for approximate solutions
Remark

• Graphical models assumed a probabilistic interpretation
  – Though they are not always learned using a probabilistic interpretation!

• The polytope view is agnostic about how you interpret the weights.
  – It only says that the decoding problem is an ILP.
3. Weighted Parsing
Grammars

• Grammars are often associated with natural language parsing, but they are extremely powerful for imposing constraints.

• We can add weights to them.
  – HMMs are a kind of weighted regular grammar (closely connected to WFSAs)
  – PCFGs are a kind of weighted CFG
  – Many, many more.

• Weighted parsing: find the \textbf{maximum-weighted derivation} for a string \(x\).
Decoding as Weighted Parsing

• Every valid y is a grammatical derivation (parse) for x.
  – HMM: sequence of “grammatical” states is one allowed by the transition table.

• Augment parsing algorithms with weights and find the best parse.

The Viterbi algorithm is an instance of recognition by a weighted grammar!
BIO Tagging as a CFG

- Weighted (or probabilistic) CKY is a dynamic programming algorithm very similar in structure to classical CKY.

4. Paths and Hyperpaths
Best Path

• General idea: take $\mathbf{x}$ and build a graph.
• Score of a path factors into the edges.

$$\arg \max_y \mathbf{w}^\top g(\mathbf{x}, y) = \arg \max_y \mathbf{w}^\top \sum_{e \in \text{Edges}} f(e) \mathbf{1}\{e \text{ is crossed by } y\text{'s path}\}$$

• Decoding is finding the best path.

The Viterbi algorithm is an instance of finding a best path!
“Lattice” View of Viterbi
A Generic Best Path Algorithm

• Input: directed graph $G = (V, E)$, cost : $E \rightarrow \mathbb{R}$, start vertex $v_0$
• Output: $d : V \rightarrow \mathbb{R}$ (shortest path function) and back pointers $b : V \rightarrow V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \emptyset$
set $d(v_0) = 0$
while $d$ has not converged:
  pick an arbitrary edge $(u, v)$
  if $d(u) + \text{cost}(u, v) < d(v)$:
    $d(v) := d(u) + \text{cost}(u, v)$
    $b(v) := u$
Ordering Updates

• Naïve ways of choosing edges will lead to cyclic updating and gross inefficiency!
• Before considering various ways of doing it, let's consider how the Viterbi algorithm is essentially solving the same problem.
Viterbi Algorithm
(In the Style of A Best Path Algorithm)

• Input:
  – directed graph $G = (V, E)$ where
    each vertex $v = (q, t), q \in Q \cup \{\varnothing\}, t \in \{-1, 0, 1, \ldots, n\}$
    and each edge $(u, v) = ((q, t), (q', t + 1))$
  – $\text{cost} : E \rightarrow \mathbb{R}$, defined by
    $\text{cost}((q, t), (q', t + 1)) = -\log \gamma(q' \mid q) - \log \eta(s_{t+1} \mid q) - \log (1 - \xi(q))$
    $\text{cost}((q, n - 1), (q', n)) = -\log \gamma(q' \mid q) - \log \eta(s_{t+1} \mid q) - \log \xi(q')$
    $\text{cost}((\varnothing, -1), (q, 0)) = -\log \pi(q)$
  – fixed start vertex $v_0 = (\varnothing, -1)$

• Output: $d : V \rightarrow \mathbb{R}$ (shortest path function) and back pointers $b : V \rightarrow V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \varnothing$
set $d(v_0) = 0$
perform a topological sort on $V$
while $d$ has not converged:
  for each $v$ in top-sort order:
    pick an arbitrary edge $(u, v)$
    for each $(u, v) \in E$:
      if $d(u) + \text{cost}(u, v) < d(v)$:
        $d(v) := d(u) + \text{cost}(u, v)$
        $b(v) := u$
      // $d(v)$ and $b(v)$ are now known
The Viterbi Trick

• From a “best path” perspective, Viterbi is:
  – defining the vertices and edges to have special structure (state/time step)
  – assigning costs based on HMM weights and the specific input string $s_1 \ldots s_n$
  – ordering the edges cleverly to make things efficient

• Note also: Viterbi's graph has no cycles.
Another Variant: “Forward” Updating

- After topological sort, can also choose all edges going out of current node.

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \emptyset$

set $d(v_0) = 0$

perform a topological sort on $V$

for each $u$ in top-sort order:

for each $(u, v) \in E$:

if $d(u) + \text{cost}(u, v) < d(v)$:

$d(v) := d(u) + \text{cost}(u, v)$

$b(v) := u$
Memoized Recursion

• Input: directed graph \( G = (V, E) \), cost \( : E \rightarrow \mathbb{R} \), start vertex \( v_0 \), target vertex \( v_t \)
• Output: \( d : V \rightarrow \mathbb{R} \) (shortest path function) and back pointers \( b : V \rightarrow V \)

for all \( v \in V \setminus \{v_0\} \), \( d(v) := \emptyset \) and \( b(v) := \emptyset \)
set \( d(v_0) = 0 \)
memoize(\( v_t \))

memoize(\( v \)):

// guaranteed to return best-cost path score for \( v \)
if \( d(v) = \emptyset \):
    \( d(v) := \infty \)
for each \( (u, v) \in E \):
    if memoize(u) + cost(u, v) < d(v):
        \( d(v) := d(u) + \text{cost}(u, v) \)
        \( b(v) := u \)
return \( d(v) \)
A Generic Best Path Algorithm

• Input: directed graph G = (V, E), cost : E → \( \mathbb{R} \), start vertex \( v_0 \)
• Output: d : V → \( \mathbb{R} \) (shortest path function) and back pointers b : V → V

for all \( v \in V \setminus \{v_0\} \), \( d(v) := \infty \) and \( b(v) := \emptyset \)
set \( d(v_0) = 0 \)
while d has not converged:
    pick an arbitrary edge (u, v)
    if \( d(u) + \text{cost}(u, v) < d(v) \):
        \( d(v) := d(u) + \text{cost}(u, v) \)
        \( b(v) := u \)
Dijkstra's Algorithm

• Input: directed graph \( G = (V, E) \), cost : \( E \to \mathbb{R}_{\geq 0} \) (important!), start vertex \( v_0 \)
• Output: \( d : V \to \mathbb{R} \) (shortest path function) and back pointers \( b : V \to V \)

for all \( v \in V \setminus \{v_0\} \), \( d(v) := \infty \) and \( b(v) := \emptyset \)
set \( d(v_0) = 0 \)
\( Q := \) priority queue on \( V \) ordered by \( d \) (lower cost = higher priority)
while \( d \) has not converged:
  while \( Q \) is not empty:
    pick an arbitrary edge \((u, v)\)
    \( u := \) extract-min \((Q)\)
    for each \((u, v) \in E:\)
      if \( d(u) + \text{cost}(u, v) < d(v)\):
        \( d(v) := d(u) + \text{cost}(u, v) \)
        \( b(v) := u \)
        update \( v \)'s priority in \( Q \)
A* Algorithm

- Input: directed graph \( G = (V, E) \), cost : \( E \rightarrow \mathbb{R}_{\geq 0} \), start vertex \( v_0 \), target vertex \( v_t \), heuristic \( h : V \rightarrow \mathbb{R}_{\geq 0} \) such that \( h(v) \leq \text{best-cost}(v, v_t) \)
- Output: \( d : V \rightarrow \mathbb{R} \) (shortest path function) and back pointers \( b : V \rightarrow V \)

for all \( v \in V \setminus \{v_0\} \), \( d(v) := \infty \) and \( b(v) := \emptyset \)
set \( d(v_0) = 0 \)
\( Q := \) priority queue on \( V \) ordered by \( d + h \) (lower cost = higher priority)
while \( Q \) is not empty:
   \( u := \text{extract-min}(Q) \)
   for each \( (u, v) \in E \):
      if \( d(u) + \text{cost}(u, v) < d(v) \):
         \( d(v) := d(u) + \text{cost}(u, v) \)
         \( b(v) := u \)
         update \( v \)'s priority in \( Q \)
Minimum Cost Hyperpath

• General idea: take $x$ and build a hypergraph.
• Score of a hyperpath factors into the hyperedges.
• Decoding is finding the best hyperpath.

• This connection was elucidated by Klein and Manning (2002).
Parsing as a Hypergraph
Parsing as a Hypergraph

cf. “Dean for democracy”
Forced to work on his thesis, sunshine streaming in the window, Mike experienced a ...
Forced to work on his thesis, sunshine streaming in the window, Mike began to ...
Why Hypergraphs?

• Useful, compact encoding of the hypothesis space.
  – Build hypothesis space using local features, maybe do some filtering.
  – Pass it off to another module for more fine-grained scoring with richer or more expensive features.
5. Weighted Logic Programming
Logic Programming

• Start with a set of **axioms** and a set of **inference rules**.

\[
\forall A, C, \quad \text{ancestor}(A, C) \iff \text{parent}(A, C) \\
\forall A, C, \quad \text{ancestor}(A, C) \iff \bigvee_B \text{ancestor}(A, B) \land \text{parent}(B, C)
\]

• The goal is to prove a specific theorem, **goal**.

• Many approaches, but we assume a **deductive** approach.
  – Start with axioms, iteratively produce more theorems.
\[
\forall \ell \in \Lambda, \quad v(\ell, 1) = \text{labeled-word}(x_1, \ell)
\]
\[
\forall \ell \in \Lambda, \quad v(\ell, i) = \bigvee_{\ell' \in \Lambda} v(\ell', i - 1) \land \text{label-bigram}(\ell', \ell) \land \text{labeled-word}(x_i, \ell)
\]
\[
\text{goal} = \bigvee_{\ell \in \Lambda} v(\ell, n)
\]
Weighted Logic Programming

• Twist: axioms have weights.
• Want the proof of goal with the best score:

$$\arg\max_y w^\top g(x, y) = \arg\max_y w^\top \sum_{a \in \text{Axioms}} f(a) \text{freq}(a; y)$$

• Note that axioms can be used more than once in a proof ($y$).
Whence WLP?

• Shieber, Schabes, and Pereira (1995): many parsing algorithms can be understood in the same deductive logic framework.

• Goodman (1999): add weights in a semiring, get many useful NLP algorithms.

Dynamic Programming

• Most views (exception is polytopes) can be understood as DP algorithms.
  – The low-level *procedures* we use are often DP.
  – Even DP is too high-level to know the best way to implement.
• Break a problem into slightly smaller problems with *optimal substructure*.
  – Best path to v depends on best paths to all u such that \((u,v) \in E\).
• Overlapping *subproblems*: each subproblem gets used repeatedly, and there aren’t too many of them.
Dynamic Programming

• Three main strategies for DP:
  – Memoization
  – Agenda (Dijkstra’s algorithm, A*)

• Things to remember in general:
  – The hypergraph may too big to represent explicitly; exhaustive calculation may be too expensive.
  – The hypergraph may or may not have properties that make “clever” orderings possible.
  – DP does not imply polynomial time and space! Most common approximations when the desired state space is too big: beam search, cube pruning, agendas with early stopping, ...
Summary

• Decoding is the general problem of choosing a complex structure.
  – Linguistic analysis, machine translation, speech recognition, ...
  – Statistical models are usually involved (not necessarily probabilistic).

• No perfect general view, but much can be gained through a combination of views.

• First question: can I solve it exactly with DP?