Dynamic Programming: Segmentation (18)

1 DYNAMIC PROGRAMMING

1.1 End-Game Plan

This week, DP examples (segmentation, LCS).
Next week, NP completeness (theory, practice).

1.2 What?

Dynamic programming is neither especially dynamic nor especially program related.
The term was coined by Richard Bellman in the late 50s to describe an algorithm-design *technique* in which an optimization problem is solved by filling in values in a table.
Various dynamic-programming algorithms have a similar flavor, but there is no single algorithm that gives you the whole story. Instead, it is usually taught by example.

1.3 DP Examples

- minimum-cost matrix-chain product: given a sequence of matrices to multiply together, what order should the intermediate products be computed to minimize the running time? (Described in book.)
- longest common subsequence: given two lists, find the largest subset of each (in order) that is the same (next time).
- longest increasing subsequence: given a list of numbers, find the largest subset (in order) that is sorted (HW).
- edit distance: given two words, how many keystrokes would it take to turn one into the other? (Useful for spelling correction and even historical linguistics, HW.)
- minimum-cost triangulation of a polygon (in book)
- TSP approximation in the plane, bitonic TSP (in book).
- shortest paths (last time, others in book).
- searching along a 1-d road

- segmentation (today)
- speech recognition: given a sequence of acoustic signals and a probabilistic model of phonemes, find the most likely phoneme (Viterbi, hidden Markov models).
- dynamic time warping: given a sequence of acoustic samples and a template model of a phoneme, find the minimum way to stretch and squeeze the sequence to fit the template.
- text alignment: given a sequence of sentences in English and their French translations and a probabilistic model relating the two languages, find the most likely way the two sequences are lined up.
- stochastic parsing: given a sentence and a stochastic grammar, find the most likely parse tree for the sentence.
- part-of-speech tagging: given a sentence and a probabilistic model of language, find the most likely part-of-speech tags for the words in the sentence.

Sequence and language related things are quite popular.

2 SEGMENTATION

2.1 Segmentation

The segmentation problem is one of taking a sequence and breaking it up into useful pieces.

```
bothearthandsaturnspin.
```

Related problems:

- Break a string of letters into words.
- Break a string of words into sentences.
- Break a string of Chinese characters into words.
- Break a sequence of phonemes into words.
- Find the most likely answer to a long crossword puzzle answer given probabilities on the crossing letters.

Greg Keim's Segmenter

2.2 Choices

Given a sequence of n characters, we need to decide where to put the spaces so that we get a sequence of valid words out.
How many different ways can we insert spaces?

> There's $n-1$ places a space can be, and each can either have a space or not... so 2^{n-1}.

On the other hand, how many possible words are in there?

> Each word must begin at some position and end at some later position, so it's something like $n(n-1)/2$.

If there are multiple ways to do this, which should we prefer?
We'll work up to a probabilistic model: choose the segmentation that gives us the most likely sequence of words.

2.3 Count in 1M Tokens (WSJ)

b	304	bo	2			both	631				
o	88										
t	4655	th	140	the	65779						
h	151	he	4602			hear	54	heart	68		
e	143			ear	7			earth	38		
a	27857			art	121						
r	207										
t	4655	th	140			than	2264				
h	151					hand	154	hands	107		
a	27857	an	4413	and	22490						
n	311	nd	6								
d	479	ds	1								
s	15889	sa	6	sat	29					saturn	1
a	27857	at	6475								
t	4655	tu	1			turn	167	turns	47		
u	2551										
r	207										
n	311										
s	15889	sp	2			spin	23				
p	464	pi	1	pin	7						
i	1666	in	23251								
n	311										
.	100000										

2.4 Algorithmic Ideas

How would you attack this?

- greedy by common words
- greedy by length
- greedy alphabetically
- path problem
- transformation rules
- probabilistic model

2.5 Simple Path Example

Want to go from beginning of sentence to the end using only valid words.
For now, let's say that valid words are 3 or more letters:

- both, the, hear, heart, ear, earth, art, than, hand, hands, and, sat, saturn, turn, turns, spin, pin

What kind of graph do we get? What is true of the order of the nodes?

DAG. Path goes left to right.

How find a path through?

DFS.

Use a table to decrease our overhead.

2.6 Problems with Path Approach

Although fast and often sufficient, there are problems with the path approach:

- Rare words: Approach will refuse to find a path with an unknown word, even if evidence is strong: "`theyattenddookuniversity.`" (Greg Keim's Segmenter).
- Competing interpretations: Approach gives no guidance if two sequence of words are possible, even if one is much more probable.

2.7 Probabilistic Language Models

A popular idea in computational linguistics is to create a probabilistic model of language. Such a model assigns a probability to every sentence in English in such a way that more likely sentences (in some sense) get higher probability. If you are unsure between two possible sentences, pick the higher probability one.
Comment: A "perfect" language model is only attainable with true intelligence. However, approximate language models are often easy to create and good enough for many applications.
Some models:

- unigram: words generated one at a time, drawn from a fixed distribution.
- bigram: probability of word depends on previous word.
- tag bigram: probability of part of speech depends on previous part of speech, probability of word depends on part of speech.
- maximum entropy: lots of other random features can contribute.
- stochastic context free: words generated by a context-free grammar augmented with probabilitistic rewrite rules.

We'll use unigrams.

2.8 Unigram Idea

Imagine that a sentence is produced by choosing a random word, or "." from a particular distribution.
Continue generating random words until a period is chosen (probability 0.1).
Picture. Expected sentence length? Mostly likely sentence? Total probability over all sentences?

> Expected length is $1/(1-p)$ for p the probability of getting a period. Most likely sentence is just ".". Total probability is 1.

2.9 Unigram Algorithms

To specify a unigram language model, we need a big dictionary that maps words to probabilities (good data structure?).

> I'm thinking hash table.

Given a unigram language model, what might we want to do?

- Generate random sentences respecting the probabilities in the model (interesting to try to solve efficiently).

Pick random words biased by the probabilities (binary search is useful here).

- Given a sentence, what is its probability?

  Just multiply the unigram probabilities together, described below.

- Given a sequence of letters, what is the most likely sentence consistent with the sequence?

  DP algorithm described below.

- Given a sequence of letters, what is the total probability of all sentences consistent with the sequence?

  DP algorithm, not described below.

2.10 Probability of a Segmentation

Example:	bo	the	art	hands	a	turns	pin	.
	2	65779	121	107	27857	47	7	100000

Probability: $156104942150629780000 0/1M^8 = 1.56 \times 10^{-27}$.
This is a million times less likely that the best sentence.

2.11 Maximum Probability Segmentation

Of all possible segmentations for a sequence of letters, we want the one with the highest probability.
Proceed like in the path example (in fact, this is a path problem, if phrased correctly...).
Start from the end... go up to "t".

2.12 Final Table

i	$s[i]$	$p[i]$	$q[i]$
1	b	1.24e-21	"both"
2	o	8.02e-25	"o"
3	t	9.11e-21	"the"
4	h	7.79e-20	"heart"
5	e	1.97e-18	"earth"
6	a	1.39e-19	"art"
7	r	5.16e-22	"r"
8	t	2.49e-18	"than"
9	h	1.14e-15	"hands"
10	a	5.17e-14	"and"
11	n	1.38e-17	"nd"
12	d	1.10e-15	"d"
13	s	2.30e-12	"saturn"
14	a	1.07e-11	"at"
15	t	3.84e-10	"turn"
16	u	3.78e-16	"u"
17	r	1.48e-13	"r"
18	n	7.15e-10	"n"
19	s	2.30e-06	"spin"
20	p	1.08e-06	"p"
21	i	2.33e-03	"in"
22	n	3.11e-05	"n"
23	.	1.00e-01	

2.13 Algorithm

Given a string of letters s and a dictionary D with (unigram) probabilities, find the probability of the most likely sequence of words.

MAXPROB(s, D)

```
 1  n ← |s|
 2  p[n + 1] ← 1.0
 3  for i ← n to 1
 4      do p[i] ← 0.0
 5          for j ← i to n
 6              do w ← s[i..j]
 7                  if D[w] · p[i − |w|] > p[i]
 8                     then p[i] = D[w] · p[i − |w|]
 9                          q[i] = w
10  return p[1]
```

2.14 Discussion

Running time: $O(n^2)$.
$p[i]$ is the probability of the most likely sequence of words in the string $s[i..n]$.
$q[i]$ is the first word in that sequence.
HW: Modify MAXPROB to print the most likely sequence of words. Give the running time for this operation.

2.15 Correctness

Write: $g : s$ to stand for all elements of the set of segmentations of a string s.
$M(s)$ is the maximum probability of a segmentation of string s.

- $M(s) = \max_{g:s} \prod_k D[g[k]]$
- $= \max_{i=1}^{|s|} \max_{g:s[(i+1)..n]} D[s[1..i]] \prod_k D[g[k]]$
- $= \max_{i=1}^{|s|} D[s[1..i]] \max_{g:s[(i+1)..n]} \prod_k D[g[k]]$
- $= \max_{i=1}^{|s|} D[s[1..i]] M(s[(i+1)..n])$

Can implement this directly to get a correct but exponential-time algorithm. Need to *reuse* results instead of recomputing over and over.
Recurrence: $T(1) = 1$, $T(n) = n + \sum_{i=1}^{n-1} T(i)$.

2.16 Algorithmic Design

- Problem: word segmentation
- Step 0 (formalize): unigram model, find maximum probability segmentation
- Step 1 (devise algorithm): table-based approach (DP)
- Step 2 (correctness): follows from equation for probability
- Step 3 (time): time per cell times number of cells ($O(n^2)$)
- refine: better formal model, better data structure, worse formal model!