Dynamic Programming and Hidden Markov Models

Geoffrey Zweig
5/14/2009
Dynamic Programming

- Probably the most important algorithmic concept
- Certainly in speech recognition
- Works by building up an optimal overall solution in terms of optimal solutions to smaller problems
  - Only applicable when problem sub-parts are independent of each other
  - Then optimally solving a sub-part without the context of everything else is OK
The Partition Problem

- Given a set of R integers > 0
- Can it be divided into two subsets such that the sum of the numbers in each subset is equal?
- Let’s call half the total sum H.
- \{5,7,4,10,3,5\} \quad H=17
  - Yes: \{(5,5,7),(10,3,4)\}
- \{5,7,1,73,18\} \quad H=52
  - No (a single number (73) is larger than H (52))
- Actually an NP-Complete problem
- But a nice illustration of DP anyway
What is the Recursion On?

- “Does a subset of the first $k$ numbers add to $x$?”
- We’ll start with $k=1$ and $x=1$
- Then work up
- And finally check what we get for $k=R$ and $x=H$
- But how to “work up”?
- Denote the $i^{th}$ number by $n(i)$.
- Key insight:
  - One can reach $x$ with the first $k$ numbers, if one has reached either $x$ or $x-n(k)$ with the first $k-1$ numbers!
- Every DP problem requires a key insight.
- The insight always comes from intuition 😊
The Recursion

- \( T(k,x) \): 1 if one can hit \( x \) with the first \( k \) numbers; else 0.
- (Ignoring initialization and boundary conditions)
  For \( k=1 \) to \( N \)
    For \( x=1 \) to \( H \)
      If \( T(k-1,x-n(k)) == 1 \) or \( T(k-1,x) == 1 \)
      then \( T(k,x) = 1 \) else \( T(k,x) = 0 \)
An Example: \{2,1,1,3,1,2\}

For \( k = 1 \) to \( N \)
    For \( x = 1 \) to \( H \)
        if \( T(k-1,x-n(k)) = 1 \) or \( T(k-1,x) = 1 \)
            then \( T(k,x) = 1 \)
        else \( T(k,x) = 0 \)

<table>
<thead>
<tr>
<th></th>
<th>x=1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n(1) = 2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Doesn’t the Order Matter?

- No.
- One can reach \( x \) with the first \( k \) numbers, if one has reached either \( x \) or \( x - n(k) \) with the first \( k-1 \) numbers!
- This is true regardless of order.
- (Try some examples at home)
Wasn’t this an NP-Complete Problem?

- Yes, and it still is
- DP works, but:
  - The size of the table is exponential in the size of the input numbers!
  - Imagine the table necessary for 64-bit input digits!
- In many useful applications, DP provides a polynomial time solution.
  - But for some problems, it is still exponential in the number or size of the inputs
Change Making

- A currency has $N$ coins with values $v(1) \ldots v(N)$
- For example, in the US, 1, 5, 10, 25 cents
- What is the minimum number of coins needed to return $X$ cents total?
- Won’t the greedy algorithm work?
  - Yes, in the US.
  - But suppose we have coins with values 1, 6, and 10, and want to hit 12.
    - Greedy would use 10 + 1 + 1, which is 3 coins
    - The optimal is 6 + 6 or 2 coins!
What will the Recursion be On?

- \( M(j,k) \): minimum number of coins it takes to get to \( k \) cents using coins 1..\( j \)

- **Key Insight:**
  - The best way of getting to \( k \) cents using all the coins up to \( j \) must either be:
    - The best way of getting there without using \( j \) or
    - The best way of getting to \( k - v(j) \) with all the coins up to (and including) \( j \), plus 1

\[
M(j,k) = \min(M(j,k - v(j)) + 1, M(j - 1, k))
\]

(Ignoring initialization and boundary conditions)
Example in a Table

- For j=1 to 3 (3 coins)
  - For k=1 to 12 (12 cents)
    - $M(j,k) = \min(M(j,k-v(j))+1, M(j-1,k))$

<table>
<thead>
<tr>
<th>K=0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>v(j)</td>
<td>0 0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Red show best way of getting to 12 cents
Blue show best way of getting to 11 cents
Segmenting an Audio File

Suppose we want to segment the audio into $k$ homogeneous segments.

Let's assume a cost function $f(a,b)$ which is high when the segment from $a$ to $b$ has dissimilar frames.
What is the best way of segmenting N frames into k blocks?

- What should the recursion be on?
- Key insight:
  - The best way of making $s$ segments ending at position $i$ has to be the best way of making $s-1$ segments that end somewhere before $i$, say at $j$, and then making a new segment, from $j+1$ to $i$.
- $C(s, i)$: The best way of making $s$ segments up to and including position $i$

Initialize:

$$C(1, i) = f(1, i) \quad i = 1..N$$

Recurse:

$$C(s, i) = \min_{j<i} C(s-1, j) + f(j+1, i) \quad s = 2..K, \ i = s..N$$
CYK Parsing

- For parsing grammars in Chomsky Normal Form
- Just two forms of rewrites:
  - A->BC \hspace{1cm} A,B,C non-terminals, or
  - A->x \hspace{1cm} A non-terminal, x terminal
- All CFGs can be turned into Chomsky Normal Form
Recursion Insight

- Consider the rule $A \rightarrow BC$
- The best way of parsing $A$ from word $i$ to word $k$ inclusive must be the best way of parsing $B$ from $i$ to some intermediate word $j$, and then the best way of parsing $C$ from $j+1$ to $k$
CYK Pseudocode

- Initialize length-1 spans with the terminal rewrites

- For each span length \( s=1..n \) (A span must be representable in terms of shorter spans)
  - For each possible starting point \( p=1..n-s+1 \)
    - The endpoint is \( e=p+s-1 \)
    - For each midpoint \( s<=m<e \)
      - For each rewrite of the form \( A -> B C \)
        - If \( A \) exists from \( s \) to \( m \), and \( B \) exists from \( m+1 \) to \( e \)
          then add \( A \) from \( s \) to \( e \)
Hidden Markov Models

Sound production modeled with the idea that we go from state to state, emitting a few feature vectors in each state, and then proceeding on.
Parts of an HMM

1) A set of states, including one start state and one final state, stop.

2) A state transition matrix $A(i,j)$, which specifies the probability of transitioning from state $i$ to state $j$.

3) An emission probability function for each state $B(j,o)$, which specifies the probability of emitting observation $o$ in state $j$.
   - This is commonly a density function, such as a gaussian

*Multiple representations are possible, e.g. with one start/end state or many
Visualizing the Transition Matrix

- A state usually has an interpretation as a part of a word – a phone or sub-phone unit
- Dictionary pronunciations impose constraints on which states should follow which
- The transition matrix is thus highly constrained and better viewed as a finite state graph

Image from Zweig et al., “Arc Minimization in Finite State Decoding Graphs...”
Visualizing a Particular Word Sequence

“A bat”

This can be seen as linearly proceeding through a sequence of states. “Beads on a string” model
Key HMM Algorithms

1) Find the probability of generating an observation sequence
2) Find the single likeliest state sequence given some observations
3) Find the parameters that maximize the data likelihood (EM)
   - This will come down to finding the a-posteriori probability of being in any state at any time
   - Note: will deal with an observation sequence from T=1..N
     - And add a dummy observation at N+1 to be consumed by the stop state.
   - Note also: everything is always conditioned on the parameters \( \Theta \) – but sometimes we leave that out to reduce clutter
Computing the Data Likelihood

We want \( P(o_1^n | \Theta) = \sum P(o_1^n, s_1^n | \Theta) = \sum P(o_1^n | s_1^n, \Theta) P(s_1^n | \Theta) \)

How can we compute this efficiently?

DP

But what is the recursion on?

Key insight:

- We can compute the probability of all the observations up to a given time and being in a given state in terms of the same quantities at the previous time.
The Alpha Recursion

\[
\alpha(i, t) = P(o_t^1, s_t = i)
\]

\[
\alpha(i, t) = \sum_j \alpha(j, t - 1)A(j, i)B(i, o_t)
\]

- Initialize \(\alpha(\text{start}, 0) = 1\)
- Define \(B(\text{stop}, N+1) = 1\), and add a dummy observation \(o_{N+1}\)
- Do the recursion from 1 to \(N+1\). Then:

\[
P(o_1^n \mid \Theta) = \alpha(\text{stop}, N + 1)
\]
To be at /ae/ at time 5, one can either be at /ae/ at time 4 and stay there, or come from /b/ at time 4.

The probabilities sum.
A Note on the Extra N+1st Observation

- It is convenient to have a single stop state.
- In ASR, though, one is often happy to match the last observation to the last state in any word.
- This would create many final states:
  - Computing the data likelihood would involve summing the alphas in all the final states.
  - Extra stuff to keep track of and do.
- So create a single final state with transitions from the word-end states.
- To avoid a distinction between emitting and non-emitting states, just add an extra “end-of-utterance” observation that only has probability (1) in the final state.
The Viterbi Algorithm

- Now we want \( \arg \max_{s_i^N} P(o_1^N, s_1^N | \Theta) = \arg \max_{s_i^N} P(o_1^N | s_1^N, \Theta)P(s_1^N | \Theta) \)

- How can we compute this efficiently?
- DP

- But what is the recursion on?

- Key insight:
  - We can compute the probability of all the observations and the likeliest state sequence up to a given time and being in a given state in terms of the same quantities at the previous time
The Delta Recursion

\[ \delta(i, t) = \arg \max_{s_{t-1}} P(o_t^t, s_{t-1}^{t-1}, s_t = i) \]

\[ \delta(i, t) = \arg \max_j \delta(j, t - 1) A(j, i) B(i, o_t) \]

- Initialize \( \delta(\text{start}, 0) = 1 \)
- Define \( B(\text{stop}, N+1) = 1 \), and add a dummy observation \( o_{N+1} \)
- Do the recursion from 1 to \( N+1 \). Then:

\[ \arg \max_{s_1^N} P(o_1^N, s_1^N | \Theta) = \delta(\text{stop}, N + 1) \]
To be at /ae/ at time 5, one can either be at /ae/ at time 4 and stay there, or come from /b/ at time 4.

Take the max.
Visualizing the Best Path

The diagram illustrates the path from /ax/ to /t/ through /b/ and /ae/. The sequence progresses from t=1 to t=7.
The Forward Backward Algorithm

- Used in EM training
- Recall that EM requires estimating complete data statistics based on the current parameters
- In the HMM case, we will need \( \gamma(i, t) = P(s_t = i \mid o_1^N) \)
- Now suppose we had \( \beta(i, t) = P(o_{t+1}^N \mid s_t = i) \)

Then:

\[
\alpha(i, t)\beta(i, t) = P(o_1^t, s_t = i)P(o_{t+1}^N \mid s_t = i) = P(o_1^N, s_t = i)
\]

So:

\[
\gamma(i, t) = P(s_t = i \mid o_1^N) = \frac{P(s_t = i, o_1^N)}{P(o_1^N)} = \frac{\alpha(i, t)\beta(i, t)}{\alpha(stop, N + 1)}
\]
Now we want: \[ \beta(i, t) = P(o_{t+1}^N \mid s_t = i) \]

How can we compute this efficiently?

But what is the recursion on?

Key insight:

- We can compute the probability of all the observations from \( t+1 \) on given that we are in state \( i \) by transitioning to another state \( j \), creating observation \( t+1 \) in \( j \) and then explaining everything from \( t+2 \) on
The Beta Recursion

\[
\beta(i,t) = P(o_{t+1}^N \mid s_t = i) \\
\beta(i,t) = \sum_j A(i,j)B(j,o_{t+1})\beta(j,t+1)
\]

- Initialize \(\beta(\text{stop}, N+1) = 1\)
- Define \(B(\text{stop}, N+1) = 1\), and add a dummy observation \(o_{N+1}\)
- Do the recursion from \(N\) down to 1

\[
\beta(i,t) = \sum_j A(i,j)B(j,o_{t+1})\beta(j,t+1) \\
\gamma(i,t) = \frac{\alpha(i,t)\beta(i,t)}{\alpha(\text{stop}, N+1)}
\]
In A Table

\[ \begin{array}{cccc}
  /ax/ & /b/ & /ae/ & /t/ \\
  t=1 & t=5 & t=7
\end{array} \]
EM Updates

- See Acero et al., Chapter 8 or Rabiner’s “Tutorial on Hidden Markov Models” for full details.
- However, with one gaussian per state:

\[
\mu_s = \frac{\sum_{t=1}^{N} \gamma(s,t) o_t}{\sum_{t=1}^{N} \gamma(s,t)}
\]

\[
\Sigma_s = \frac{\sum_{t=1}^{N} \gamma(s,t)(o_t - \mu_s)(o_t - \mu_s)'}{\sum_{t=1}^{N} \gamma(s,t)}
\]
Question to Think About

- Suppose we want an estimate of how many times we have transitioned from state $i$ to state $j$
- What would that be?
Break

- Next:
- Context Sensitivity
- Organizing a simple trainer and decoder
- Bayesian Network Representations
Transitions

\[
P(o_1^N, s_t = x, s_{t+1} = y) \\
= P(o_1^t, s_t = x)A(x, y)B(y, o_{t+1})P(o_{t+2}^N | s_{t+1} = y) \\
= \alpha(x, t)A(x, y)B(y, o_{t+1})\beta(y, t + 1)
\]
Context Sensitivity

- Consider “An eye” and “The icicle”
  - ae n ay
  - dh iy ay s ih k ax l
- The phoneme ay might sound different in the two cases because of the preceding phone.
  - Co-articulatory effects
- Is an /ay/ really an /ay/?
- How can we determine what units to use?
Possible Phonetic Units

- **Monophones** – the regular phones, ~40
  - `dh iy ay s ih k ax l`
- **Biphones** - two phone sequences ~1600
  - `dh_iy iy_ay ay_s s_ih ih_k k_ax ax_l l_*`
- **Triphones** – three phone sequences ~64,000
  - `_dh_iy dh_iy_ay iy_ay_s ay_s_ih s_ih_k ih_k_ax k_ax_l ax_l_*`

  The longer the span, the more accurately we can hope to model co-articulatory effects.

  - But the more units we need to model
    - Will they all have enough training data?
    - Are we also creating some useless units that sound the same as others, anyway?

  - Can we intelligently tie subsets of units together in equivalence classes?
Data Driven Clustering

- Build unclustered models
- Then merge similar ones in a bottom-up fashion
- For example, distance between clusters might be KL divergence, or the average probability of component means wrt the other model

Fig. 10.2 Data-driven state tying

From S. Young et al., HTKBook
Decision Trees

- Suppose we model all the data with a gaussian. That has some likelihood.
- Now consider splitting on a phonetic question.
  - There will be a gain in likelihood by modeling the two parts with different gaussians.
- Recursively split the data on the question that gives the biggest likelihood gain.
- Stop when there are too few examples.
- Key advantage:
  - Unseen phone sequences have well-defined models.

From S. Young et al., HTKBook
Recognizing Continuous Digits: Training

- Few enough words that we can model each word with its own unique state sequence
- “one” -> 1_1, 1_2, 1_3 ... 1_n
- “two” -> 2_1, 2_2, 2_3 ... 2_n
- Associate a self-loop probability and forward-jump probability with each state (beads-on-a-string model)
- Training needs to consider linear state sequences, e.g. “two one two”
  - Make an appropriate state graph for each training utterance
  - Each state in this graph will be labeled with an underlying HMM state
  - The transition probabilities will reflect the underlying self-loop / forward jump probabilities
- Posterior probabilities are computed for each state in the graph, and then the statistics accumulated for the underlying HMM states
Training Example

“One Two One”
Assume 3 states per word

Graph states are numbered 1..9

“one, state 2” model, accumulator
But under-the-covers, the models and accumulators are tied
Both for observation and transition probabilities!
Recognizing Continuous Digits: Decoding

- Each word has a linear sequence of states
- The end of each word has a transition to the beginning of every other word
  - A bigram language model is implicitly encoded in the transition probabilities
- Encode the whole graph in a transition matrix \( A \)
- Run the Viterbi algorithm
Scaling Up: The Key Issue

- Representing allowable state sequences
  - Language model provides probability distribution over word sequences
    - Big but sparse
  - Dictionary provides constraints
    - Some phone sequences never seen
  - Decision tree maps context independent sequences into context-sensitive sequences

- Basic idea:
  - For each training sequence we’ll create a little graph that represents the allowable transitions.
    - Then we’ll do forward-backward with that and accumulate statistics
  - To decode, we create a giant graph representing all possible word sequences at the state level.
    - Then we’ll apply the Viterbi algorithm to recover the likeliest state and word sequence.
Incorporating the Dictionary and Decision Tree

- This is easy if there is no cross-word acoustic context
- The state sequence for each word can be created once
- Then we can do “search-and-replace” as necessary
- We will not discuss cross-word context
Representing the Language Model

Build the LM graph, then insert the states

Glossed-over issue: We have introduced non-emitting States.
Finite State Representations

- The state-sequencing constraints can also be represented as a graph where the arcs represent the emitting HMM states
- When this is done, there are some nice algorithms for manipulating the graphs
  - Determinizing them (good for pruning characteristics at decode time)
    - This is the really important one
  - Minimizing them
**Finite State Acceptors (FSAs)**

- Graphs with start state, final state
- Arcs are labeled with symbols
- A sequence $S$ is accepted if a path from the start state to the final state matches the symbols in $S$

![Diagram of a finite state acceptor](image)

Accepts “I want it” and “I’d like it” but not “I’d want it”

An edge can be added from the final state back to the start state to allow for longer sequences.
Weighted Finite State Acceptors

- Same, except that each edge has a cost
- The cost of accepting a sequence is the sum of the edge costs
- When edge costs are negative log-probabilities, this is useful
Determinization

- A FSA is determinized if no two arcs emanating from the same state have the same label
  - (And there are no epsilon arcs)

Pruning in conjunction with the Viterbi algorithm is often more effective when using determinized representations
Minimization

- Minimizing a deterministic FSA produces a deterministic FSA that is equivalent, and has the smallest number of states of all equivalent deterministic FSAs.

Finite State Machines in Speech and Language Processing

- More general set of representations and operations defined and championed by Mohri, Riley & Pereira at AT&T
- Check out: http://www.research.att.com/~fsmtools/fsm/
Scaling Up: A Summary

- Encapsulate the language model, dictionary and decision tree by using a finite state representation of the HMM states and possible transitions
- Determinization and Minimization can be useful in making this representation compact and efficient to compute with
- Training involves running forward-backward on a separate graph for each training sentence
- Decoding involves running Viterbi on a graph that encapsulates the full language model
Bayesian Network Representations

\[ P(A,B,C,D,E) = P(A) P(B|A) P(C|A) P(D|B,C) P(E|C) \]

BN factors probability distribution with chain rule and conditional independence assumptions
Bayesian Network Algorithms

- Find the data probability:
  - Sum over all assignments of values to the hidden variables $v$ of $P(v,o)$
  - Analogous to HMM sum over hidden state sequences
- Find the likeliest assignment of values to the variables $v$ given the observations
  - Analogous to finding the single likeliest state sequence for HMMs (Viterbi)
- Find the parameters that maximize the data likelihood
  - Analogous to HMM EM process
- For details, see Zweig, “Bayesian Network Structures and Inference Techniques for Automatic Speech Recognition”, CSL 2003.
Bayesian Network Structures for ASR

Homework

- Write a program to compute the minimum edit distance between two strings
  - An edit is an insertion, deletion or substitution and counts as 1.
  - Matches count 0 (they are free).
- The minimum edit distance is the minimum number of edits needed to turn the reference into the hypothesis
- Operations:
  - Match a word in the reference to a word in the hypothesis (cost 0)
  - Substitute a word in the reference for a word in the hypothesis (cost 1)
  - Delete a word in the reference (cost 1)
  - Insert a word in the reference (cost 1)
- Example of (usually) non-minimal edit distance:
  - Delete everything in the reference
  - Insert everything in the hypothesis
Reference: the dog ran down the street
Hypothesis: a the dog runs the street

3 edits necessary:
- Insert “a”
- Delete “down”
- Substitute ran/runs
Homework: Process

- Write the core alignment function
  - This function should compute the edit distance
  - And output the actual operations used
- Write a wrapper that reads a sequence of hypotheses from one file, and a sequence of reference strings from another file, and computes the edit distance for each hypothesis/reference pair
- Apply this to the strings in the files “samples.hyp” and “samples.ref” provided on the web page
- Turn in:
  - A copy of the program
  - The total number of edits needed across all strings (summed over all strings – this is a “checksum”).
  - The detailed alignments of the first 10 strings.