Dynamic Programming

Avishek Dutta

CS155 Machine Learning and Data Mining

February 14, 2017

Motivation

Much of machine learning is heavily dependent on computational power

Many libraries exist that aim to reduce computational time

- TensorFlow
- Spark

Well-designed algorithms also speed up computation

Dynamic Programming

Dynamic Programming is a programming technique that leverages previous computations

Can be applied when a problem has the following properties:

- optimal sub-structure
- overlapping sub-problems

Examples include:

- Fibonacci Numbers
- Viterbi Algorithm
- Forward/Backward Algorithm

Optimal Substructure

What does it mean for a problem to exhibit the optimal substructure property?

Solution to optimization problem can be solved by combining optimal solutions to subproblems

Example

Mergesort and Quicksort both display the optimal substructure property

Overlapping Subproblem

What does it mean for a problem to exhibit the overlapping subproblem property?

Subproblems are solved repeatedly to obtain the optimal solution to the main optimization problem

Example

Mergesort and Quicksort do not display the overlapping subproblems property

Fibonacci Numbers

Write a function to find the n-th Fibonacci number.

$$F_n = F_{n-1} + F_{n-2}$$

where $F_2 = 1$ and $F_1 = 1$

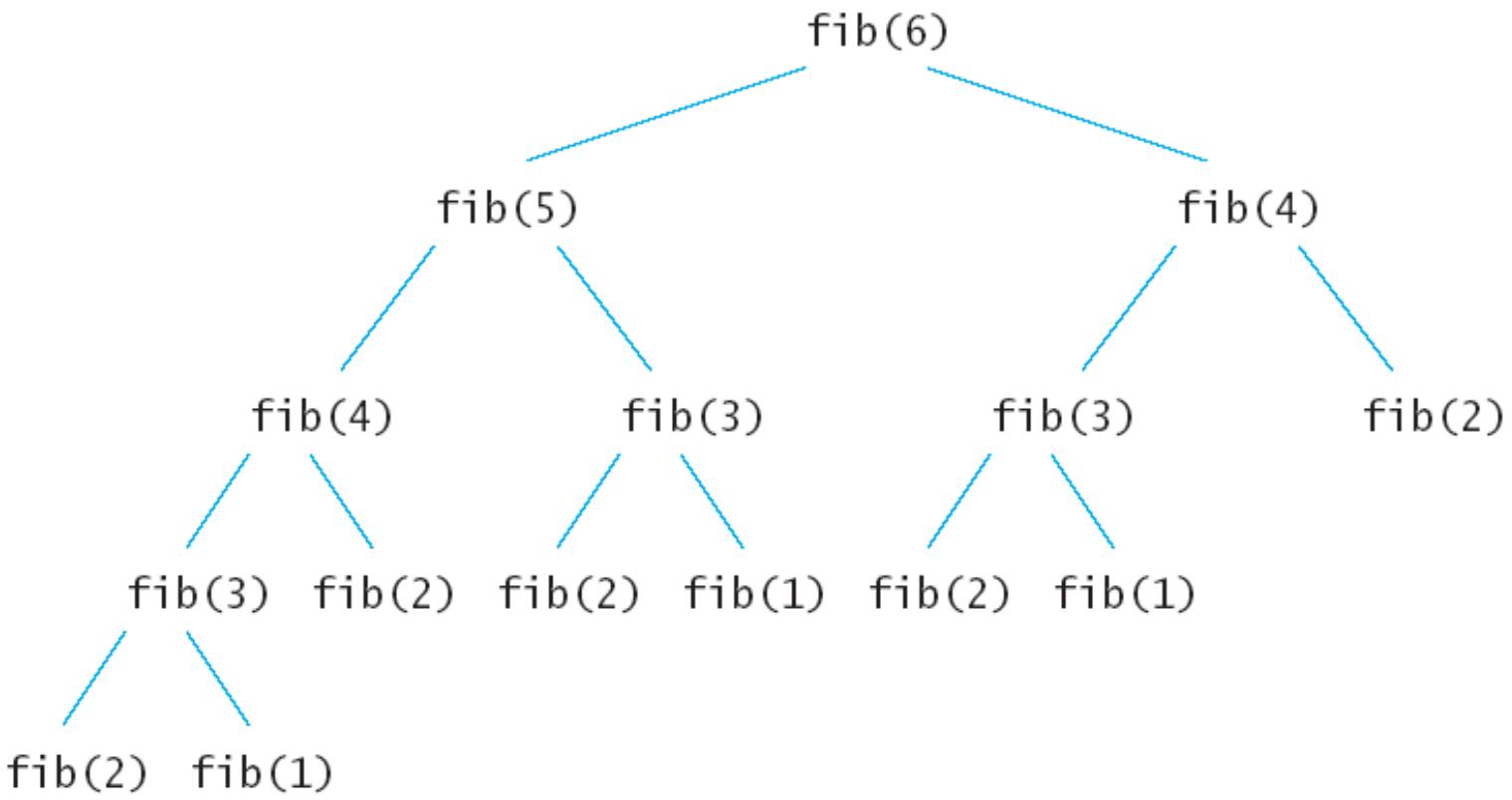
This is easy right?

Example

```
def naive_fib(n):
  if n == 1 or n == 2:
    return 1
  else:
    return naive_fib(n-1) + naive_fib(n-2)
```

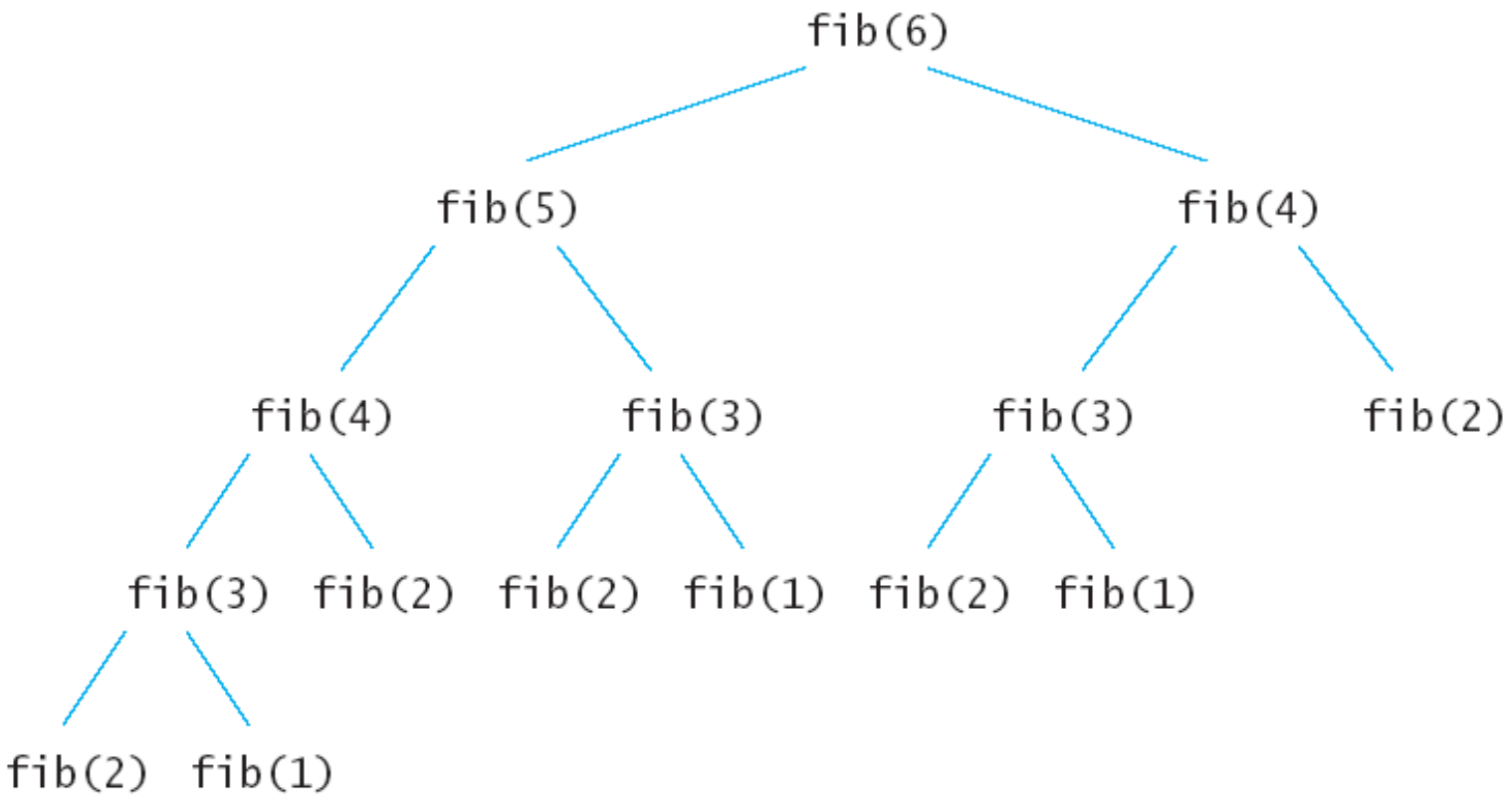
Fibonacci Numbers

How does the computation break down?



Fibonacci Numbers

Overlapping subproblems? Optimal substructure?



Fibonacci Numbers

So we can use Dynamic Programming. But how?

Two main approaches:

- Top-down
- Bottom-up

Top-down: solve recursively, storing previous computations for later use

Bottom-up: build a table of subproblem results that grows until we reach solution

Top-down Fibonacci Numbers

Recursively solve, storing results of subproblems as we go

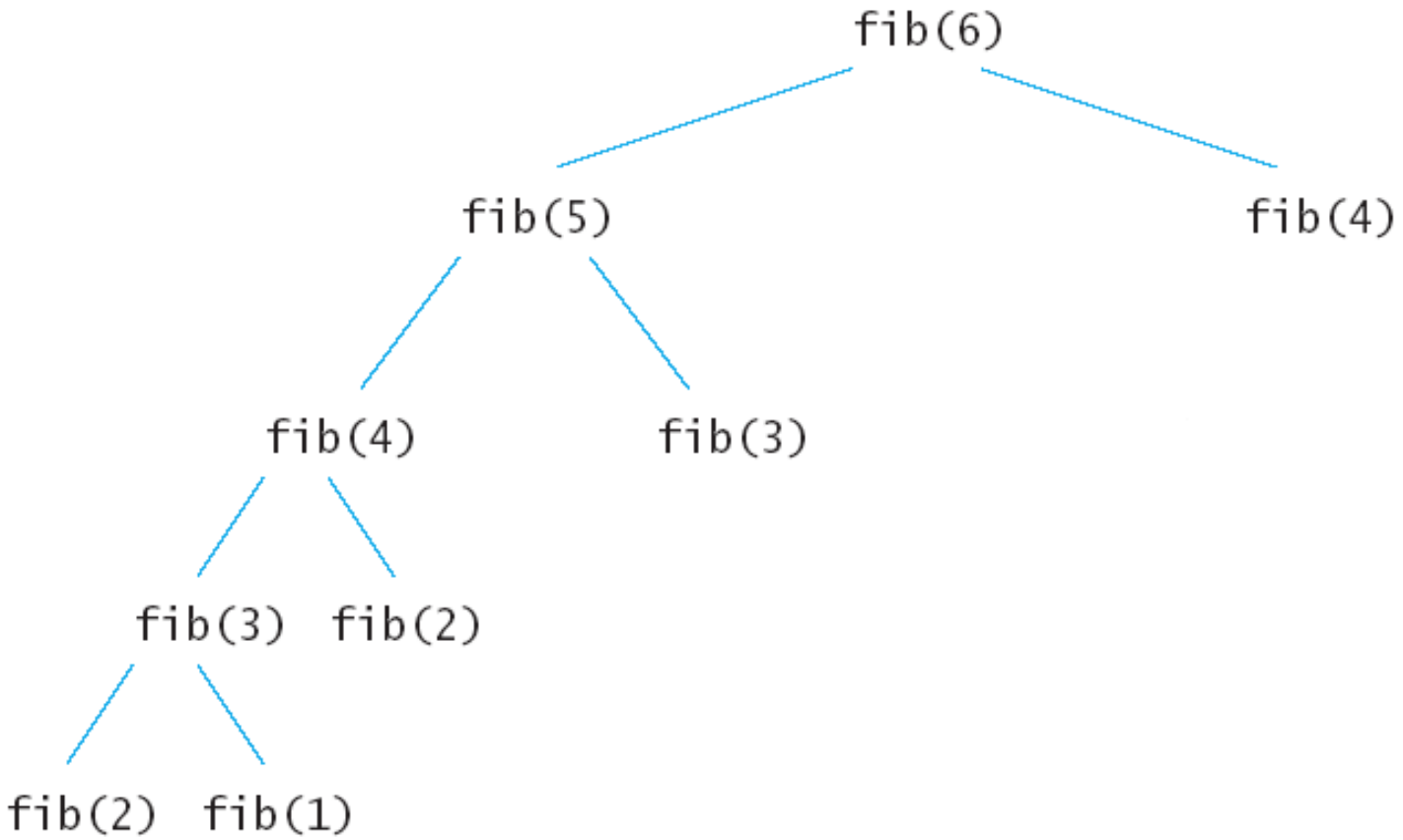
Example

```
table = {}

def top_down_fib(n):
  if n in table:
    return table[n]
  else:
    if n == 1 or n == 2:
      table[n] = 1
    else:
      table[n] = top_down_fib(n-1) + top_down_fib(n-2)
    return table[n]
```

Top-down Fibonacci Numbers

What's the computation path?



Bottom-up Fibonacci Numbers

Build a table of subproblem results, starting from the base cases

Example

```
def bottom_up_fib(n):
  if n == 1 or n == 2:
    return 1
  else:
    table = [0, 1, 1]
    for i in range(3, n+1):
      table.append(table[i-1] + table[i-2])
    return table[n]
```

Bottom-up Fibonacci Numbers

What's the computation path?

fib(1)	fib(2)	fib(3)	fib(4)	fib(5)	fib(6)
1	1	2	3	5	8

Fibonacci Numbers

What does this accomplish?

Reduces the number of computations and overall time complexity

$$\mathcal{O}(2^n) \rightarrow \mathcal{O}(n)$$

Dramatic speedup, especially for large n

Viterbi Algorithm

Fibonacci numbers are easy - what about a harder problem? Let's shift gears and talk about HMMs

Recall that with a 1st order HMM

$$P(x, y) = P(End \mid y^M) \prod_{i=1}^{M} P(y^i \mid y^{i-1}) \prod_{i=1}^{M} P(x^i \mid y^i)$$

- $P(x^i \mid y^i) \rightarrow$ probability of state y^i generating emission x^i
- $P(y^i \mid y^{i-1}) \rightarrow$ probability of state y^{i-1} transitioning to y^i
- $P(y^1 \mid y^0) \rightarrow$ probability of the start state
- $P(End \mid y^M) \rightarrow$ optional

Viterbi Algorithm

$$P(x, y) = P(End \mid y^M) \prod_{i=1}^{M} P(y^i \mid y^{i-1}) \prod_{i=1}^{M} P(x^i \mid y^i)$$

Suppose we have a length M sequence of emissions, x. How can we find the length M sequence of states, y, for which $P(x, y)$ is maximized?

Consider the naive solution: generate all possible sequences y.

How many sequences are there? L^M if there are L possible states.

This is too slow. Can we do better? Of course!

Dynamic Programming for Viterbi Algorithm

We know that Dynamic Programming can be applied here, but why and how?

The problem has the optimal substructure and overlapping subproblem properties.

To understand this, lets move to a more concrete example

Suppose that x is a sentence and y is the corresponding part-of-speech (POS) tag sequence.

$$y^i \in L = \{N = Noun, V = Verb, D = Adverb\}$$

Dynamic Programming for Viterbi Algorithm

$$L = \{N = Noun, V = Verb, D = Adverb\}$$

Original Problem: Given sentence, $x^{1:M}$, find the sequence, $\hat{y}^M$, of POS tags that maximizes $P(x, \hat{y}^M)$

Equivalent Problem: Given sentence, $x^{1:M}$, find L sequences, $\hat{y}^M$, of POS tags that maximize $P(x, \hat{y}^M)$, one of each ending in $\{N, V, D\}$

$$\hat{y}^M(N) = y^1 y^2 \dots y^{M-1} N$$
$$\hat{y}^M(V) = y^1 y^2 \dots y^{M-1} V$$
$$\hat{y}^M(D) = y^1 y^2 \dots y^{M-1} D$$

Optimal Substructure in Viterbi Algorithm

Subproblem: Given length $M-1$ sentence, $x^{1:M-1}$, find L sequences, $\hat{y}^{M-1}$, of POS tags that maximize $P(x^{1:M-1}, \hat{y}^{M-1})$, one of each ending in $\{N, V, D\}$

$$\hat{y}^{M-1}(N) = y^1 y^2 \dots y^{M-2} N$$
$$\hat{y}^{M-1}(V) = y^1 y^2 \dots y^{M-2} V$$
$$\hat{y}^{M-1}(D) = y^1 y^2 \dots y^{M-2} D$$

Can we use the optimal solution to this subproblem to solve the overall problem? Yes!

Optimal Substructure in Viterbi Algorithm

Optimal solutions to subproblems:

$$S = \left\{ \hat{y}^{M-1}(N), \hat{y}^{M-1}(V), \hat{y}^{M-1}(D) \right\}$$

Optimal solution to overall problem:

$$\hat{y}^{M}(N) = \left\{ \arg\max_{y \in S} P(x^{1:M}, y \oplus N) \right\} \oplus N$$

$$\hat{y}^{M}(V) = \left\{ \arg\max_{y \in S} P(x^{1:M}, y \oplus V) \right\} \oplus V$$

$$\hat{y}^{M}(D) = \left\{ \arg\max_{y \in S} P(x^{1:M}, y \oplus D) \right\} \oplus D$$

where $\oplus$ is concatenation

Optimal Substructure in Viterbi Algorithm

Why does this give us the optimal solution?

$$\hat{y}^M(N) = \left\{ \arg\max_{y \in S} P(x^{1:M}, y \oplus N) \right\} \oplus N$$

Because of the structure of the model:

$$P(x^{1:M-1}, y^{1:M-1}) = \prod_{i=1}^{M-1} P(y^i \mid y^{i-1}) \prod_{i=1}^{M-1} P(x^i \mid y^i)$$

$$P(x^{1:M}, y^{1:M}) = \prod_{i=1}^{M} P(y^i \mid y^{i-1}) \prod_{i=1}^{M} P(x^i \mid y^i)$$

$$P(x^{1:M}, y^{1:M}) = P(x^{1:M-1}, y^{1:M-1}) P(y^M \mid y^{M-1}) P(x^M \mid y^M)$$

Optimal Substructure in Viterbi Algorithm

More formally, suppose that $\hat{y}^M(N)$ was formed using some $y \notin S$

$$S = \left\{ \hat{y}^{M-1}(N), \hat{y}^{M-1}(V), \hat{y}^{M-1}(D) \right\}$$

This y must end in one of $\{N, V, D\}$. Since the structure of the model is:

$$P(x^{1:M}, y^{1:M}) = P(x^{1:M-1}, y^{1:M-1}) P(y^M \mid y^{M-1}) P(x^M \mid y^M)$$

We can replace y with the correct element from S to get better $\hat{y}^M(N)$.

Overlapping Subproblems in Viterbi Algorithm

Equivalent Problem: Given sentence, $x^{1:M}$, find L sequences, $\hat{y}^M$, of POS tags that maximize $P(x, \hat{y}^M)$, one of each ending in $\{N, V, D\}$

Our problem has the optimal substructure property. What about the overlapping subproblem property? Easy to see:

$$\hat{y}^M(N) = \left\{ \arg\max_{y \in S} P(x^{1:M}, y \oplus N) \right\} \oplus N$$

$$\hat{y}^M(V) = \left\{ \arg\max_{y \in S} P(x^{1:M}, y \oplus V) \right\} \oplus V$$

$$\hat{y}^M(D) = \left\{ \arg\max_{y \in S} P(x^{1:M}, y \oplus D) \right\} \oplus D$$

Dynamic Programming in Viterbi Algorithm

Now we understand why we can use dynamic programming for this problem. But how do we do it?

Use a bottom-up approach. Build a table of solutions to the suproblems. Extend the table until we have $\hat{y}^M(N)$, $\hat{y}^M(V)$, $\hat{y}^M(D)$.

How do we start? Trivially:

$$\hat{y}^1(N) = N$$
$$\hat{y}^1(V) = V$$
$$\hat{y}^1(D) = D$$

Dynamic Programming in Viterbi Algorithm

$$\hat{y}^i(t) = \left\{ \underset{y_s \in S_i}{\arg\max}\ P(x^{1:i}, y_s \oplus N) \right\} \oplus t$$

$$= \left\{ \underset{y_s \in S_i}{\arg\max}\ P(x^{1:i-1}, y_s) P(y^i = t \mid y^{i-1} = s) P(x^i \mid y^i = t) \right\} \oplus t$$

	1	2	...	M-1	M
N	N	VN	...	...N	...VN
V	V	VV	...	...V	...DV
D	D	ND	...	...D	...DD

$$S_i = \{y_n = \hat{y}^{i-1}(N), y_v = \hat{y}^{i-1}(V), y_d = \hat{y}^{i-1}(D)\}$$

$$t \in \{N, V, D\}$$

Forward/Backward Algorithm

Now let's shift gears and talk about the Forward/Backward Algorithm

For unsupervised training of HMMs, we need to be able to compute the terms

$$P(y^i = z \mid x) \text{ and } P(y^i = b, y^{i-1} = a \mid x)$$

These expressions can be written in terms of $\alpha_z(i)$ and $\beta_z(i)$

$$\alpha_z(i) = P(x^{1:i}, y^i = z \mid A, O)$$
$$\beta_z(i) = P(x^{i+1:M} \mid y^i = z)$$

Forward Algorithm

Problem: Compute $\alpha_z(i) = P(x^{1:i}, y^i = z \mid A, O)$ for all z, i.

Naive solution: sum over all possible sequences $y^{1:i-1}$:

$$\alpha_z(i) = \sum_{y^{1:i-1}} P(x^{1:i}, y^i = z, y^{1:i-1} \mid A, O)$$

This is too slow. Can we apply dynamic programming here? Yes!

Let's see how the $\alpha_z(i)$ terms exhibit optimal substructure and overlapping subproblems.

Optimal Substructure in Forward Algorithm

Suppose that x is a sentence and y is the corresponding part-of-speech (POS) tag sequence.

$$y^i \in L = \{N, V, D\}$$

'Optimal' solutions for a given subproblem:

$$\alpha_n^i = P(x^i \mid y^i = n) \sum_{j \in L} \alpha_j^{i-1} P(y^i = n \mid y^{i-1} = j)$$

$$\alpha_v^i = P(x^i \mid y^i = v) \sum_{j \in L} \alpha_j^{i-1} P(y^i = v \mid y^{i-1} = j)$$

$$\alpha_d^i = P(x^i \mid y^i = d) \sum_{j \in L} \alpha_j^{i-1} P(y^i = d \mid y^{i-1} = j)$$

Remember: $\alpha_j^{i-1} = P(x^{1:i-1}, y^{i-1} = j \mid A, 0)$

Optimal Substructure in Forward Algorithm

Why are can we build α_j^i from α_j^{i-1}?

The law of total probability!

$$\begin{aligned}\alpha_j^i &= P(x^{1:i}, y^i = j \mid A, 0) \\ &= \sum_{k \in L} P(x^{1:i}, y^i = j, y^{i-1} = k \mid A, 0) \\ &= \sum_{K \in L} \alpha_k^{i-1} P(y^i = j \mid y^{i-1} = k) P(x^i \mid y^i = j)\end{aligned}$$

Overlapping Subproblems in Forward Algorithm

As in the Viterbi Algorithm, the overlapping subproblems is easy to see here as well

$$\alpha_n^i = P(x^i \mid y^i = n) \sum_{j \in L} \alpha_j^{i-1} P(y^i = n \mid y^{i-1} = j)$$

$$\alpha_v^i = P(x^i \mid y^i = v) \sum_{j \in L} \alpha_j^{i-1} P(y^i = v \mid y^{i-1} = j)$$

$$\alpha_d^i = P(x^i \mid y^i = d) \sum_{j \in L} \alpha_j^{i-1} P(y^i = d \mid y^{i-1} = j)$$

Remember: $\alpha_j^{i-1} = P(x^{1:i-1}, y^{i-1} = j \mid A, 0)$

Dynamic Programming in Forward Algorithm

Now we've confirmed that we can use dynamic programming for this problem. But how do we do it?

Again, use a bottom-up approach. Build a table of solutions to subproblems. In other words, build a table of $\alpha_z(i)$ values for all z, i

How do we start? Trivially:

$$\alpha_n^1 = P(x^{1:1}, y^1 = n \mid A, O) = P(x^1 \mid y^1 = n)P(y^1 = n \mid y^0)$$
$$\alpha_v^1 = P(x^{1:1}, y^1 = v \mid A, O) = P(x^1 \mid y^1 = v)P(y^1 = v \mid y^0)$$
$$\alpha_d^1 = P(x^{1:1}, y^1 = d \mid A, O) = P(x^1 \mid y^1 = d)P(y^1 = d \mid y^0)$$

Proceed from here using the equations from the previous page

Backward Algorithm

Problem: Compute $\beta_z(i) = P(x^{i+1:M} \mid y^i = z, A, O)$ for all z, i.

Naive solution: sum over all possible sequences $y^{i+1:M}$:

$$\beta_z(i) = \sum_{y^{i+1:M}} P(x^{i+1:M}, y^{i+1:M} \mid y^i = z, A, O)$$

This is too slow. Can we apply dynamic programming here? Yes!

Let's see how the $\beta_z(i)$ terms exhibit optimal substructure and overlapping subproblems.

Optimal Substructure in Backward Algorithm

Suppose that x is a sentence and y is the corresponding part-of-speech (POS) tag sequence.

$$y^i \in L = \{N, V, D\}$$

'Optimal' solutions for a given subproblem:

$$\beta_n^i = \sum_{j \in L} \beta_j^{i+1} P(y^{i+1} = j \mid y^i = n) P(x^{i+1} \mid y^{i+1} = j)$$

$$\beta_v^i = \sum_{j \in L} \beta_j^{i+1} P(y^{i+1} = j \mid y^i = v) P(x^{i+1} \mid y^{i+1} = j)$$

$$\beta_d^i = \sum_{j \in L} \beta_j^{i+1} P(y^{i+1} = j \mid y^i = d) P(x^{i+1} \mid y^{i+1} = j)$$

Remember: $\beta_j^{i+1} = P(x^{i+2:M} \mid y^{i+1} = j, A, O)$

Overlapping Subproblems in Backward Algorithm

As in the Viterbi Algorithm, the overlapping subproblems is easy to see here as well

$$\beta_n^i = \sum_{j \in L} \beta_j^{i+1} P(y^{i+1} = j \mid y^i = n) P(x^{i+1} \mid y^{i+1} = j)$$

$$\beta_v^i = \sum_{j \in L} \beta_j^{i+1} P(y^{i+1} = j \mid y^i = v) P(x^{i+1} \mid y^{i+1} = j)$$

$$\beta_d^i = \sum_{j \in L} \beta_j^{i+1} P(y^{i+1} = j \mid y^i = d) P(x^{i+1} \mid y^{i+1} = j)$$

Remember: $\beta_j^{i+1} = P(x^{i+2:M} \mid y^{i+1} = j, A, O)$

Dynamic Programming in Backward Algorithm

Now we've confirmed that we can use dynamic programming for this problem. But how do we do it?

Again, use a bottom-up approach (sort of). Build a table of solutions to subproblems. In other words, build a table of $\beta_z(i)$ values for all z, i

How do we start? Trivially:

$$\beta_n^M = P(x^{M+1:M} \mid y^M = n, A, O) = 1$$
$$\beta_v^M = P(x^{M+1:M} \mid y^M = v, A, O) = 1$$
$$\beta_d^M = P(x^{M+1:M} \mid y^M = d, A, O) = 1$$

Proceed **backward** using the equations from the previous slides