Dynamic Programming

Ellen Feldman and Avishek Dutta

CS155 Machine Learning and Data Mining

February 27, 2018

Motivation

Much of machine learning is heavily dependent on computational power

Many libraries exist that aim to reduce computational time

- TensorFlow
- Spark

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 is a programming/algorithmic technique that leverages previous computations to save computation time

Dynamic Programming

Dynamic Programming is a programming/algorithmic technique that leverages previous computations to save computation time

Can be applied when a problem has the following properties:

- Optimal substructure
- Overlapping subproblems

Dynamic Programming

Dynamic Programming is a programming/algorithmic technique that leverages previous computations to save computation time

Can be applied when a problem has the following properties:

- Optimal substructure
- Overlapping subproblems

Examples include:

- Fibonacci Numbers
- Viterbi Algorithm
- Forward/Backward Algorithm

Aside: why the name 'Dynamic Programming'?

From Richard Bellman's autobiography:

Aside: why the name ‘Dynamic Programming’?

From Richard Bellman’s autobiography:

I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, ‘Where did the name, dynamic programming, come from?’ The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research...His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially.

Aside: why the name 'Dynamic Programming'?

Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. ... I decided therefore to use the word, 'programming.' I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying—I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

Dynamic Programming

Dynamic Programming is a programming/algorithmic technique that leverages previous computations to save computation time

Can be applied when a problem has the following properties:

- Optimal substructure
- Overlapping subproblems

Examples include:

- Fibonacci Numbers
- Viterbi Algorithm
- Forward/Backward Algorithm

Optimal Substructure

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

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

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

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

Optimal substructure is associated with recursion and divide-and-conquer strategies

Overlapping Subproblems

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

Problem can be broken down into subproblems which use each other's results

Overlapping Subproblems

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

Problem can be broken down into subproblems which use each other's results

Subproblems' results are reused repeatedly to solve the main optimization problem

Overlapping Subproblems

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

Problem can be broken down into subproblems which use each other's results

Subproblems' results are reused repeatedly to solve the main optimization problem

Example

Mergesort and Quicksort do not display the overlapping subproblems property

Relationship between Optimal Substructure, Overlapping Subproblems, and Dynamic Programming

Note: Optimal substructure is a pre-condition for the overlapping subproblems property!

Relationship between Optimal Substructure, Overlapping Subproblems, and Dynamic Programming

Note: Optimal substructure is a pre-condition for the overlapping subproblems property!

- Optimal substructure: obtain optimal solution by combining optimal solutions to subproblems
  - Divide-and-conquer approach, e.g. Mergesort

Relationship between Optimal Substructure, Overlapping Subproblems, and Dynamic Programming

Note: Optimal substructure is a pre-condition for the overlapping subproblems property!

- Optimal substructure: obtain optimal solution by combining optimal solutions to subproblems
  - Divide-and-conquer approach, e.g. Mergesort
- Overlapping subproblems: subproblems reuse each other's results in the computation process
  - E.g. Fibonacci, Viterbi, forward/backward algorithms

Relationship between Optimal Substructure, Overlapping Subproblems, and Dynamic Programming

Note: Optimal substructure is a pre-condition for the overlapping subproblems property!

- Optimal substructure: obtain optimal solution by combining optimal solutions to subproblems
  - Divide-and-conquer approach, e.g. Mergesort
- Overlapping subproblems: subproblems reuse each other's results in the computation process
  - E.g. Fibonacci, Viterbi, forward/backward algorithms
- Dynamic programming: store and leverage previous computations to save computation time

Relationship between Optimal Substructure, Overlapping Subproblems, and Dynamic Programming

Note: Optimal substructure is a pre-condition for the overlapping subproblems property!

- Optimal substructure: obtain optimal solution by combining optimal solutions to subproblems
  - Divide-and-conquer approach, e.g. Mergesort
- Overlapping subproblems: subproblems reuse each other's results in the computation process
  - E.g. Fibonacci, Viterbi, forward/backward algorithms
- Dynamic programming: store and leverage previous computations to save computation time
  - Would not be helpful for optimal substructure-only case—overkill!

Relationship between Optimal Substructure, Overlapping Subproblems, and Dynamic Programming

Note: Optimal substructure is a pre-condition for the overlapping subproblems property!

- Optimal substructure: obtain optimal solution by combining optimal solutions to subproblems
  - Divide-and-conquer approach, e.g. Mergesort
- Overlapping subproblems: subproblems reuse each other's results in the computation process
  - E.g. Fibonacci, Viterbi, forward/backward algorithms
- Dynamic programming: store and leverage previous computations to save computation time
  - Would not be helpful for optimal substructure-only case—overkill!
  - With overlapping subproblems: can reduce runtime from exponential to polynomial time

Fibonacci Numbers: An Example

Let's see an example of where the optimal substructure and overlapping subproblems properties are helpful.

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

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

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

Fibonacci Numbers: An Example

Let's see an example of where the optimal substructure and overlapping subproblems properties are helpful.

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

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

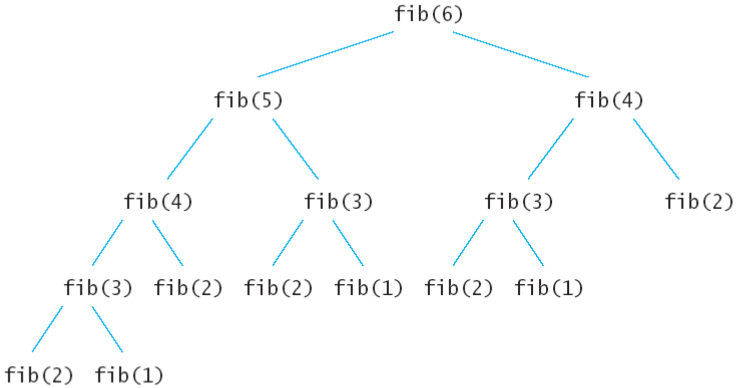
where $F_2 = 1$ and $F_1 = 1$

Example

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

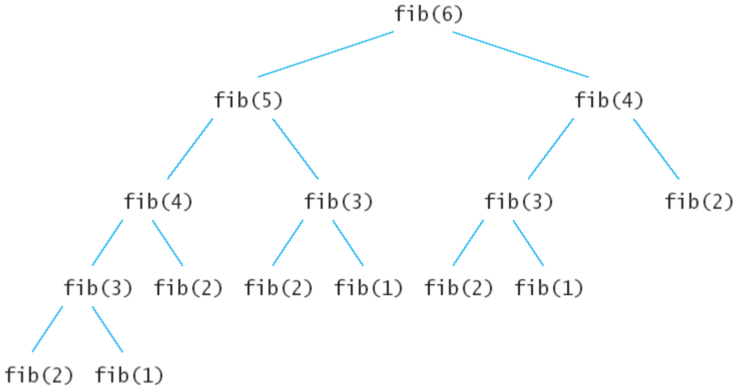
Fibonacci Numbers

How does the computation break down?



Fibonacci Numbers

Optimal substructure? Overlapping subproblems?



Fibonacci Numbers

So we can use Dynamic Programming.

Two main approaches for implementing Dynamic Programming:

- Top-down
- Bottom-up

Fibonacci Numbers

So we can use Dynamic Programming.

Two main approaches for implementing Dynamic Programming:

- Top-down
- Bottom-up

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

Fibonacci Numbers

So we can use Dynamic Programming.

Two main approaches for implementing Dynamic Programming:

- Top-down
- Bottom-up

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

Bottom-up: build a table of subproblem results (starting with base cases) that grows until we reach solution

Top-down Fibonacci Numbers

Recursively solve, storing results of subproblems as we go

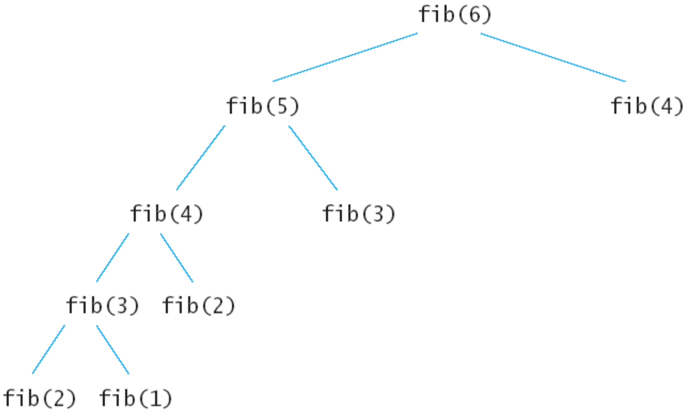
Example

```
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

What did dynamic programming accomplish here?

Fibonacci Numbers

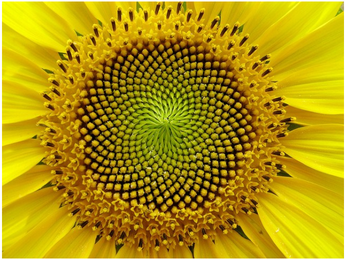
What did dynamic programming accomplish here?

Reduces the number of computations and overall time complexity

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

Dramatic speedup, especially for large n

Fun fact: Fibonacci numbers show up in nature!



Reference: Bohannon, John. “Sunflowers Show Complex Fibonacci Sequences.” News from *Science*, 2016, `http://www.sciencemag.org/news/2016/05/sunflowers-show-complex-fibonacci-sequences`.

Hidden Markov Models

Now, let's see how dynamic programming helps us with HMMs

Now, let's see how dynamic programming helps us with HMMs

Recall that with a 1st-order HMM

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

Hidden Markov Models

Now, let's see how dynamic programming helps us with HMMs

Recall that with a 1st-order HMM

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

- $P(x^i \mid y^i) \rightarrow$ 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

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

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

Viterbi Algorithm

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

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

Consider the naive solution:
$\mathbf{y}^* = \arg\max_{\mathbf{y}} \log P(\mathbf{y} \mid \mathbf{x}) = \arg\max_{\mathbf{y}} \log P(\mathbf{x}, \mathbf{y})$.

Viterbi Algorithm

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

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

Consider the naive solution:
$\mathbf{y}^* = \arg\max_{\mathbf{y}} \log P(\mathbf{y} \mid \mathbf{x}) = \arg\max_{\mathbf{y}} \log P(\mathbf{x}, \mathbf{y})$.

This requires evaluating L^M sequences if there are L possible states.

This is too slow. Can we do better?

Dynamic Programming for Viterbi Algorithm

Can we use Dynamic Programming to find the most probable state sequence $\mathbf{y}$? Why?

Dynamic Programming for Viterbi Algorithm

Can we use Dynamic Programming to find the most probable state sequence $\mathbf{y}$? Why?

This problem has the optimal substructure and overlapping subproblem properties.

Dynamic Programming for Viterbi Algorithm

Can we use Dynamic Programming to find the most probable state sequence $\mathbf{y}$? Why?

This problem has the optimal substructure and overlapping subproblem properties.

To understand this, let's move to a more concrete example

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

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

Dynamic Programming for Viterbi Algorithm

$$S = \{N = Noun, V = Verb, D = Adverb\}, L = |S| = 3$$

Dynamic Programming for Viterbi Algorithm

$$S = \{N = Noun, V = Verb, D = Adverb\}, L = |S| = 3$$

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

Dynamic Programming for Viterbi Algorithm

$$S = \{N = Noun, V = Verb, D = Adverb\}, L = |S| = 3$$

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

Approach: Let $\hat{\mathbf{y}}_a^j$ be the most-probable length-j sequence ending in state $a \in S$.

Dynamic Programming for Viterbi Algorithm

$$S = \{N = Noun, V = Verb, D = Adverb\}, L = |S| = 3$$

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

Approach: Let $\hat{\mathbf{y}}_a^j$ be the most-probable length-j sequence ending in state $a \in S$.
Find L sequences $\hat{\mathbf{y}}_a^M$ of POS tags maximizing $P(\mathbf{x}, \hat{\mathbf{y}}_a^M)$:

$$\hat{\mathbf{y}}_N^M = y^1 y^2 \ldots y^{M-1} N$$
$$\hat{\mathbf{y}}_V^M = y^1 y^2 \ldots y^{M-1} V$$
$$\hat{\mathbf{y}}_D^M = y^1 y^2 \ldots y^{M-1} D$$

Dynamic Programming for Viterbi Algorithm

$$S = \{N = Noun, V = Verb, D = Adverb\}, L = |S| = 3$$

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

Approach: Let $\hat{\mathbf{y}}_a^j$ be the most-probable length-j sequence ending in state $a \in S$.
Find L sequences $\hat{\mathbf{y}}_a^M$ of POS tags maximizing $P(\mathbf{x}, \hat{\mathbf{y}}_a^M)$:

$$\hat{\mathbf{y}}_N^M = y^1 y^2 \dots y^{M-1} N$$
$$\hat{\mathbf{y}}_V^M = y^1 y^2 \dots y^{M-1} V$$
$$\hat{\mathbf{y}}_D^M = y^1 y^2 \dots y^{M-1} D$$

Then, $\mathbf{y}^* = \underset{\mathbf{y} = \hat{\mathbf{y}}_a^M, a \in S}{\arg\max} \; P(\mathbf{x}, \mathbf{y})$

Optimal Substructure in Viterbi Algorithm

Problem: Find $\hat{\mathbf{y}}_N^M$, $\hat{\mathbf{y}}_V^M$, and $\hat{\mathbf{y}}_D^M$

Optimal Substructure in Viterbi Algorithm

Problem: Find $\hat{\mathbf{y}}_N^M$, $\hat{\mathbf{y}}_V^M$, and $\hat{\mathbf{y}}_D^M$

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

$$\hat{\mathbf{y}}_N^{M-1} = y^1 y^2 \ldots y^{M-2} N$$
$$\hat{\mathbf{y}}_V^{M-1} = y^1 y^2 \ldots y^{M-2} V$$
$$\hat{\mathbf{y}}_D^{M-1} = y^1 y^2 \ldots y^{M-2} D$$

Optimal Substructure in Viterbi Algorithm

Problem: Find $\hat{\mathbf{y}}_N^M$, $\hat{\mathbf{y}}_V^M$, and $\hat{\mathbf{y}}_D^M$

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

$$\hat{\mathbf{y}}_N^{M-1} = y^1 y^2 \dots y^{M-2} N$$
$$\hat{\mathbf{y}}_V^{M-1} = y^1 y^2 \dots y^{M-2} V$$
$$\hat{\mathbf{y}}_D^{M-1} = y^1 y^2 \dots y^{M-2} D$$

How can we use the optimal solution to this subproblem to solve the problem stated above?

Optimal Substructure in Viterbi Algorithm

Optimal solutions to subproblems:

$$\left\{\hat{\mathbf{y}}_N^{M-1}, \hat{\mathbf{y}}_V^{M-1}, \hat{\mathbf{y}}_D^{M-1}\right\} = \left\{\hat{\mathbf{y}}_a^{M-1} | a \in S\right\}$$

Optimal Substructure in Viterbi Algorithm

Optimal solutions to subproblems:

$$\left\{\hat{\mathbf{y}}_N^{M-1}, \hat{\mathbf{y}}_V^{M-1}, \hat{\mathbf{y}}_D^{M-1}\right\} = \left\{\hat{\mathbf{y}}_a^{M-1} | a \in S\right\}$$

Optimal solution to overall problem:

$$\hat{\mathbf{y}}_N^M = \left\{\underset{\mathbf{y} \in \hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} P(\mathbf{x}^{1:M}, \mathbf{y} \oplus N)\right\} \oplus N$$

$$\hat{\mathbf{y}}_V^M = \left\{\underset{\mathbf{y} \in \hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} P(\mathbf{x}^{1:M}, \mathbf{y} \oplus V)\right\} \oplus V$$

$$\hat{\mathbf{y}}_D^M = \left\{\underset{\mathbf{y} \in \hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} P(\mathbf{x}^{1:M}, \mathbf{y} \oplus D)\right\} \oplus D$$

where $\oplus$ is concatenation

Optimal Substructure in Viterbi Algorithm

Why does this give us the optimal solution?

$$\hat{\mathbf{y}}_N^M = \left\{ \underset{\mathbf{y} \in \hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} \; P(\mathbf{x}^{1:M}, \mathbf{y} \oplus N) \right\} \oplus N$$

Optimal Substructure in Viterbi Algorithm

Why does this give us the optimal solution?

$$\hat{\mathbf{y}}_N^M = \left\{ \underset{\mathbf{y} \in \hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} \; P(\mathbf{x}^{1:M}, \mathbf{y} \oplus N) \right\} \oplus N$$

Because of the structure of the model:

Optimal Substructure in Viterbi Algorithm

Why does this give us the optimal solution?

$$\hat{\mathbf{y}}_N^M = \left\{ \underset{\mathbf{y} \in \hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} \; P(\mathbf{x}^{1:M}, \mathbf{y} \oplus N) \right\} \oplus N$$

Because of the structure of the model:

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

Optimal Substructure in Viterbi Algorithm

Why does this give us the optimal solution?

$$\hat{\mathbf{y}}_N^M = \left\{ \underset{\mathbf{y} \in \hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} \; P(\mathbf{x}^{1:M}, \mathbf{y} \oplus N) \right\} \oplus N$$

Because of the structure of the model:

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

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

Optimal Substructure in Viterbi Algorithm

Why does this give us the optimal solution?

$$\hat{\mathbf{y}}_N^M = \left\{ \underset{\mathbf{y} \in \hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} \; P(\mathbf{x}^{1:M}, \mathbf{y} \oplus N) \right\} \oplus N$$

Because of the structure of the model:

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

$$\begin{aligned} P(\mathbf{x}^{1:M}, \mathbf{y}^{1:M}) &= \prod_{i=1}^{M} P(y^i \mid y^{i-1}) \prod_{i=1}^{M} P(x^i \mid y^i) \\ &= P(\mathbf{x}^{1:M-1}, \mathbf{y}^{1:M-1}) P(y^M \mid y^{M-1}) P(x^M \mid y^M) \end{aligned}$$

Optimal Substructure in Viterbi Algorithm

Prove optimal substructure by contradiction: suppose that $\hat{\mathbf{y}}_N^M$ was formed using some length-(M - 1) $\mathbf{y} \notin \{\hat{\mathbf{y}}_a^{M-1} | a \in S\}$

Optimal Substructure in Viterbi Algorithm

Prove optimal substructure by contradiction: suppose that $\hat{\mathbf{y}}_N^M$ was formed using some length-(M - 1) $\mathbf{y} \notin \{\hat{\mathbf{y}}_a^{M-1} | a \in S\}$

This $\mathbf{y}$ must end in some $a \in S = \{N, V, D\}$. Without loss of generality, assume $\mathbf{y}$ ends in N.

Optimal Substructure in Viterbi Algorithm

Prove optimal substructure by contradiction: suppose that $\hat{\mathbf{y}}_N^M$ was formed using some length-(M - 1) $\mathbf{y} \notin \{\hat{\mathbf{y}}_a^{M-1} | a \in S\}$

This $\mathbf{y}$ must end in some $a \in S = \{N, V, D\}$. Without loss of generality, assume $\mathbf{y}$ ends in N.

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

Optimal Substructure in Viterbi Algorithm

Prove optimal substructure by contradiction: suppose that $\hat{\mathbf{y}}_N^M$ was formed using some length-(M - 1) $\mathbf{y} \notin \{\hat{\mathbf{y}}_a^{M-1} | a \in S\}$

This $\mathbf{y}$ must end in some $a \in S = \{N, V, D\}$. Without loss of generality, assume $\mathbf{y}$ ends in N.

$$\begin{aligned} P(\mathbf{x}^{1:M}, \mathbf{y}^{1:M}) &= P(\mathbf{x}^{1:M-1}, \mathbf{y}^{1:M-1}) P(y^M \mid y^{M-1}) P(x^M \mid y^M) \\ &\leq P(\mathbf{x}^{1:M-1}, \hat{\mathbf{y}}_N^{M-1}) P(y^M \mid y^{M-1}) P(x^M \mid y^M) \end{aligned}$$

Optimal Substructure in Viterbi Algorithm

Prove optimal substructure by contradiction: suppose that $\hat{\mathbf{y}}_N^M$ was formed using some length-(M - 1) $\mathbf{y} \notin \{\hat{\mathbf{y}}_a^{M-1} | a \in S\}$

This $\mathbf{y}$ must end in some $a \in S = \{N, V, D\}$. Without loss of generality, assume $\mathbf{y}$ ends in N.

$$\begin{aligned} P(\mathbf{x}^{1:M}, \mathbf{y}^{1:M}) &= P(\mathbf{x}^{1:M-1}, \mathbf{y}^{1:M-1}) P(y^M \mid y^{M-1}) P(x^M \mid y^M) \\ &\leq P(\mathbf{x}^{1:M-1}, \hat{\mathbf{y}}_N^{M-1}) P(y^M \mid y^{M-1}) P(x^M \mid y^M) \end{aligned}$$

So, we can replace $\mathbf{y}$ with $\hat{\mathbf{y}}_a^{M-1}$ to get better (i.e. more probable) $\hat{\mathbf{y}}_N^M$.

Overlapping Subproblems in Viterbi Algorithm

Our problem has the optimal substructure property. We can also see that it has the overlapping subproblems property:

$$\hat{\mathbf{y}}_N^M = \left\{ \underset{\mathbf{y}=\hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} \; P(\mathbf{x}^{1:M}, \mathbf{y} \oplus N) \right\} \oplus N$$

$$\hat{\mathbf{y}}_V^M = \left\{ \underset{\mathbf{y}=\hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} \; P(\mathbf{x}^{1:M}, \mathbf{y} \oplus V) \right\} \oplus V$$

$$\hat{\mathbf{y}}_D^M = \left\{ \underset{\mathbf{y}=\hat{\mathbf{y}}_a^{M-1}, a \in S}{\arg\max} \; P(\mathbf{x}^{1:M}, \mathbf{y} \oplus D) \right\} \oplus D$$

Dynamic Programming in Viterbi Algorithm

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

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 $\hat{\mathbf{y}}_N^M$, $\hat{\mathbf{y}}_V^M$, $\hat{\mathbf{y}}_D^M$.

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 $\hat{\mathbf{y}}_N^M$, $\hat{\mathbf{y}}_V^M$, $\hat{\mathbf{y}}_D^M$.

How do we start?

$$\hat{\mathbf{y}}_N^1 = N$$
$$\hat{\mathbf{y}}_V^1 = V$$
$$\hat{\mathbf{y}}_D^1 = D$$

Dynamic Programming in Viterbi Algorithm

a is some part of speech, $a \in S = \{N, V, D\}$

$$\begin{aligned}
\hat{\mathbf{y}}^i_a &= \left\{ \operatorname*{arg\,max}_{\mathbf{y}=\hat{\mathbf{y}}^{i-1}_a, a \in S} P(\mathbf{x}^{1:i}, \mathbf{y} \oplus a) \right\} \oplus a \\
&= \left\{ \operatorname*{arg\,max}_{\mathbf{y}=\hat{\mathbf{y}}^{i-1}_a, a \in S} P(\mathbf{x}^{1:i-1}, \mathbf{y}) P(y^i = a \mid y^{i-1}) P(x^i \mid y^i = a) \right\} \oplus a
\end{aligned}$$

Dynamic Programming in Viterbi Algorithm

a is some part of speech, $a \in S = \{N, V, D\}$

$$\begin{aligned}
\hat{\mathbf{y}}_a^i &= \left\{ \operatorname*{arg\,max}_{\mathbf{y}=\hat{\mathbf{y}}_a^{i-1}, a \in S} P(\mathbf{x}^{1:i}, \mathbf{y} \oplus a) \right\} \oplus a \\
&= \left\{ \operatorname*{arg\,max}_{\mathbf{y}=\hat{\mathbf{y}}_a^{i-1}, a \in S} P(\mathbf{x}^{1:i-1}, \mathbf{y}) P(y^i = a \mid y^{i-1}) P(x^i \mid y^i = a) \right\} \oplus a
\end{aligned}$$

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

Applications of the Viterbi Algorithm

- Computational linguistics, natural language processing
  - Part-of-speech tagging

Applications of the Viterbi Algorithm

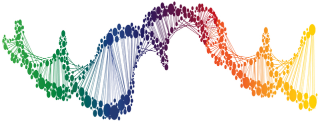
- Computational linguistics, natural language processing
  - Part-of-speech tagging
  - Speech recognition, synthesis, and enhancement algorithms

Applications of the Viterbi Algorithm

- Computational linguistics, natural language processing
  - Part-of-speech tagging
  - Speech recognition, synthesis, and enhancement algorithms
- Telecommunications
  - Cell phones, radio, satellites, wireless networks

Applications of the Viterbi Algorithm

- Computational linguistics, natural language processing
  - Part-of-speech tagging
  - Speech recognition, synthesis, and enhancement algorithms
- Telecommunications
  - Cell phones, radio, satellites, wireless networks
- Bioinformatics
  - Genome sequencing, modeling families of proteins



Computing Marginal Probabilities: Forward/Backward Algorithm

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

Computing Marginal Probabilities: 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,

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

Computing Marginal Probabilities: 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,

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

These expressions can be written in terms of $\alpha_a(i)$ and $\beta_b(i)$:

$$\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a)$$
$$\beta_b(i) \propto P(\mathbf{x}^{i+1:M} \mid y^i = b)$$

Forward Algorithm

Problem: Compute $\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a)$ for all a, i.

Forward Algorithm

Problem: Compute $\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a)$ for all a, i.

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

$$\alpha_a(i) \propto \sum_{\mathbf{y}^{1:i-1}} P(\mathbf{x}^{1:i}, y^i = a, \mathbf{y}^{1:i-1})$$

Forward Algorithm

Problem: Compute $\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a)$ for all a, i.

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

$$\alpha_a(i) \propto \sum_{\mathbf{y}^{1:i-1}} P(\mathbf{x}^{1:i}, y^i = a, \mathbf{y}^{1:i-1})$$

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

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

Optimal Substructure in Forward Algorithm

First, let's see how to build $\alpha_a(i)$ from $\alpha_a(i-1)$

Recall: $y^i \in S = \{N, V, D\}$

Optimal Substructure in Forward Algorithm

First, let's see how to build $\alpha_a(i)$ from $\alpha_a(i-1)$

Recall: $y^i \in S = \{N, V, D\}$

$\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a)$

Optimal Substructure in Forward Algorithm

First, let's see how to build $\alpha_a(i)$ from $\alpha_a(i-1)$

Recall: $y^i \in S = \{N, V, D\}$

$$\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a) = \sum_{a' \in S} P(\mathbf{x}^{1:i}, y^i = a, y^{i-1} = a')$$

Optimal Substructure in Forward Algorithm

First, let's see how to build $\alpha_a(i)$ from $\alpha_a(i-1)$

Recall: $y^i \in S = \{N, V, D\}$

$$\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a) = \sum_{a' \in S} P(\mathbf{x}^{1:i}, y^i = a, y^{i-1} = a')$$
$$= \sum_{a' \in S} P(\mathbf{x}^{1:i-1}, y^{i-1} = a') P(y^i = a \mid y^{i-1} = a') P(x^i \mid y^i = a)$$

Optimal Substructure in Forward Algorithm

First, let's see how to build $\alpha_a(i)$ from $\alpha_a(i-1)$

Recall: $y^i \in S = \{N, V, D\}$

$$\begin{aligned}
\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a) &= \sum_{a' \in S} P(\mathbf{x}^{1:i}, y^i = a, y^{i-1} = a') \\
&= \sum_{a' \in S} P(\mathbf{x}^{1:i-1}, y^{i-1} = a') P(y^i = a \mid y^{i-1} = a') P(x^i \mid y^i = a) \\
&\propto \sum_{a' \in S} \alpha_{a'}(i-1) P(y^i = a \mid y^{i-1} = a') P(x^i \mid y^i = a)
\end{aligned}$$

Optimal Substructure in Forward Algorithm

First, let's see how to build $\alpha_a(i)$ from $\alpha_a(i-1)$

Recall: $y^i \in S = \{N, V, D\}$

$$\alpha_a(i) \propto P(\mathbf{x}^{1:i}, y^i = a) = \sum_{a' \in S} P(\mathbf{x}^{1:i}, y^i = a, y^{i-1} = a')$$
$$= \sum_{a' \in S} P(\mathbf{x}^{1:i-1}, y^{i-1} = a') P(y^i = a \mid y^{i-1} = a') P(x^i \mid y^i = a)$$
$$\propto \sum_{a' \in S} \alpha_{a'}(i-1) P(y^i = a \mid y^{i-1} = a') P(x^i \mid y^i = a)$$
$$\alpha_a(i) = \sum_{a' \in S} \alpha_{a'}(i-1) P(y^i = a \mid y^{i-1} = a') P(x^i \mid y^i = a)$$

Optimal Substructure in Forward Algorithm

Remember: $\alpha_a(i-1) \propto P(\mathbf{x}^{1:i-1}, y^{i-1} = a)$

'Optimal' solutions for the subproblems:

$$\alpha_N(i) = P(x^i \mid y^i = N) \sum_{a \in S} \alpha_a(i-1) P(y^i = N \mid y^{i-1} = a)$$
$$\alpha_V(i) = P(x^i \mid y^i = V) \sum_{a \in S} \alpha_a(i-1) P(y^i = V \mid y^{i-1} = a)$$
$$\alpha_D(i) = P(x^i \mid y^i = D) \sum_{a \in S} \alpha_a(i-1) P(y^i = D \mid y^{i-1} = a)$$

Overlapping Subproblems in Forward Algorithm

Remember: $\alpha_a(i-1) \propto P(\mathbf{x}^{1:i-1}, y^{i-1} = a)$

As in the Viterbi Algorithm, we can see the overlapping subproblems here as well:

$$\alpha_N(i) = P(x^i \mid y^i = N) \sum_{a \in S} \alpha_a(i-1) P(y^i = N \mid y^{i-1} = a)$$

$$\alpha_V(i) = P(x^i \mid y^i = V) \sum_{a \in S} \alpha_a(i-1) P(y^i = V \mid y^{i-1} = a)$$

$$\alpha_D(i) = P(x^i \mid y^i = D) \sum_{a \in S} \alpha_a(i-1) P(y^i = D \mid y^{i-1} = a)$$

Dynamic Programming in Forward Algorithm

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

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, i.e. a table of $\alpha_a(i)$ values for all a, i

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, i.e. a table of $\alpha_a(i)$ values for all a, i

How do we start?

$$\alpha_N(1) = P(x^{1:1}, y^1 = N) = P(x^1 \mid y^1 = N)P(y^1 = N \mid y^0)$$
$$\alpha_V(1) = P(x^{1:1}, y^1 = V) = P(x^1 \mid y^1 = V)P(y^1 = V \mid y^0)$$
$$\alpha_D(1) = P(x^{1:1}, y^1 = D) = P(x^1 \mid y^1 = D)P(y^1 = D \mid y^0)$$

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, i.e. a table of $\alpha_a(i)$ values for all a, i

How do we start?

$$\alpha_N(1) = P(x^{1:1}, y^1 = N) = P(x^1 \mid y^1 = N)P(y^1 = N \mid y^0)$$
$$\alpha_V(1) = P(x^{1:1}, y^1 = V) = P(x^1 \mid y^1 = V)P(y^1 = V \mid y^0)$$
$$\alpha_D(1) = P(x^{1:1}, y^1 = D) = P(x^1 \mid y^1 = D)P(y^1 = D \mid y^0)$$

Proceed from here using the equations from the previous page

Backward Algorithm

Problem: Compute $\beta_b(i) \propto P(\mathbf{x}^{i+1:M} \mid \mathbf{y}^i = b)$ for all b, i.

Backward Algorithm

Problem: Compute $\beta_b(i) \propto P(\mathbf{x}^{i+1:M} \mid \mathbf{y}^i = b)$ for all b, i.

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

$$\beta_b(i) \propto \sum_{\mathbf{y}^{i+1:M}} P(\mathbf{x}^{i+1:M}, \mathbf{y}^{i+1:M} \mid y^i = b)$$

Problem: Compute $\beta_b(i) \propto P(\mathbf{x}^{i+1:M} \mid \mathbf{y}^i = b)$ for all b, i.

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

$$\beta_b(i) \propto \sum_{\mathbf{y}^{i+1:M}} P(\mathbf{x}^{i+1:M}, \mathbf{y}^{i+1:M} \mid y^i = b)$$

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

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

Optimal Substructure in Backward Algorithm

Remember: $\beta_b(i+1) \propto P(\mathbf{x}^{i+2:M} \mid y^{i+1} = b)$

Similarly to the $\alpha_a(i)$ values, we can also recursively define $\beta_b(i)$:

$$\beta_b(i) = \sum_{b' \in S} \beta_{b'}(i+1) P(y^{i+1} = b' \mid y^i = b) P(x^{i+1} \mid y^{i+1} = b')$$

Optimal Substructure in Backward Algorithm

Remember: $\beta_b(i+1) \propto P(\mathbf{x}^{i+2:M} \mid y^{i+1} = b)$

Similarly to the $\alpha_a(i)$ values, we can also recursively define $\beta_b(i)$:

$$\beta_b(i) = \sum_{b' \in S} \beta_{b'}(i+1) P(y^{i+1} = b' \mid y^i = b) P(x^{i+1} \mid y^{i+1} = b')$$

'Optimal' solutions for the subproblems:

$$\beta_N(i) = \sum_{b \in S} \beta_b(i+1) P(y^{i+1} = b \mid y^i = N) P(x^{i+1} \mid y^{i+1} = b)$$

$$\beta_V(i) = \sum_{b \in S} \beta_b(i+1) P(y^{i+1} = b \mid y^i = V) P(x^{i+1} \mid y^{i+1} = b)$$

$$\beta_D(i) = \sum_{b \in S} \beta_b(i+1) P(y^{i+1} = b \mid y^i = D) P(x^{i+1} \mid y^{i+1} = b)$$

Overlapping Subproblems in Backward Algorithm

Remember: $\beta_b(i+1) \propto P(\mathbf{x}^{i+2:M} \mid y^{i+1} = b)$

As in the Viterbi and forward algorithms, we can see the overlapping subproblems property:

$$\beta_N(i) = \sum_{b \in S} \beta_b(i+1) P(y^{i+1} = b \mid y^i = N) P(x^{i+1} \mid y^{i+1} = b)$$

$$\beta_V(i) = \sum_{b \in S} \beta_b(i+1) P(y^{i+1} = b \mid y^i = V) P(x^{i+1} \mid y^{i+1} = b)$$

$$\beta_D(i) = \sum_{b \in S} \beta_b(i+1) P(y^{i+1} = b \mid y^i = D) P(x^{i+1} \mid y^{i+1} = b)$$

Dynamic Programming in Backward Algorithm

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

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. Build a table of solutions to subproblems, i.e. a table of $\beta_b(i)$ values for all b, i

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. Build a table of solutions to subproblems, i.e. a table of $\beta_b(i)$ values for all b, i

How do we start?

$$\beta_N(M) = P(\mathbf{x}^{M+1:M} \mid y^M = N) = 1$$
$$\beta_V(M) = P(\mathbf{x}^{M+1:M} \mid y^M = V) = 1$$
$$\beta_D(M) = P(\mathbf{x}^{M+1:M} \mid y^M = D) = 1$$

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. Build a table of solutions to subproblems, i.e. a table of $\beta_b(i)$ values for all b, i

How do we start?

$$\beta_N(M) = P(\mathbf{x}^{M+1:M} \mid y^M = N) = 1$$
$$\beta_V(M) = P(\mathbf{x}^{M+1:M} \mid y^M = V) = 1$$
$$\beta_D(M) = P(\mathbf{x}^{M+1:M} \mid y^M = D) = 1$$

Initialize as 1, then proceed **backward** using the equations from the previous slides

Thanks for joining!

Questions?