Introduction to
Dynamic Programming for
Generative Models

Andreas Stuhlmüller
IPAM GSS 2011
Overview

- Motivation
  - A simple coordination game
  - Blue-eyed islanders puzzle
  - Hidden Markov models
- DP for HMMs: Forward and Viterbi algorithm
- DP for probabilistic programs: UDP algorithm
A simple coordination game

(define (sample-location)
  (if (flip .55)
      'popular-bar
      'unpopular-bar))

(define (alice)
  (query
   (define alice-location (sample-location))
   (equal? alice-location (bob (- depth 1))))

(define (bob)
  (sample-location))
A simple coordination game

(define (sample-location)
  (if (flip .55)
      'popular-bar
      'unpopular-bar))

(define (alice)
  (query
   (define alice-location (sample-location))
   alice-location
   (equal? alice-location (bob)))))

(define (bob)
  (sample-location))

(Schelling 1960)
A simple coordination game

(define (sample-location)
  (if (flip .55)
      'popular-bar
      'unpopular-bar))

(define (alice depth)
  (query
    (define alice-location (sample-location))
    alice-location
    (equal? alice-location (bob (- depth 1)))))

(define (bob depth)
  (query
    (define bob-location (sample-location))
    bob-location
    (or (= depth 0)
        (equal? bob-location (alice depth))))

(Schelling 1960)
A simple coordination game

(define (sample-location)
  (if (flip .55)
      'popular-bar
      'unpopular-bar))

(define (alice depth)
  (query
   (define alice-location (sample-location))
   alice-location
   (equal? alice-location (bob (- depth 1))))))

(define (bob depth)
  (query
   (define bob-location (sample-location))
   bob-location
   (or (= depth 0)
       (equal? bob-location (alice depth)))))))

(Schelling 1960)
The blue-eyed islanders puzzle

(Moses, Dolev, and Halpern, 1986; Tao, 2008)
The blue-eyed islanders puzzle

(Moses, Dolev, and Halpern, 1986; Tao, 2008)
The blue-eyed islanders puzzle

At least one of you has blue eyes.

(Moses, Dolev, and Halpern, 1986; Tao, 2008)
The blue-eyed islanders puzzle

- Two arguments:
  - “The outsider provides no new information, therefore nothing changes.”
  - “By induction on the number of blue-eyed people, all $N$ blue-eyed people leave on day $N$. ”
The blue-eyed islanders puzzle

- Difficult for current universal inference algorithms
  - Deeply nested queries
  - Complex (almost-)deterministic dependencies
- But: Potential for sharing of subcomputations
The blue-eyed islanders puzzle
The blue-eyed islanders puzzle
The blue-eyed islanders puzzle
Markov Models

- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix $A$
- Distribution on initial states $P_{init}$
- Set of end states $Q_{end}$
Markov Models

- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix $A$
- Distribution on initial states $P_{init}$
- Set of end states $Q_{end}$

Markov assumption:

$$P(q_i|q_1, \ldots, q_{i-1}) = P(q_i|q_{i-1})$$
Markov Models

- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix $A$
- Distribution on initial states $P_{init}$
- Set of end states $Q_{end}$

Slide:

- So-so
- So-so
- Interesting
- Interesting
- So-so
Markov Models

- Set of states $Q = \{S_+, S_0, S_\text{-}\}$
- Transition probability matrix $A$
- Distribution on initial states $P_{init}$
- Set of end states $Q_{end}$

$$P([S_0, S_0, S_+, S_+, S_0]) =$$

$$P_{init}(S_0) \times P(S_0|S_0) \times P(S_+|S_0) \times P(S_+|S_+) \times P(S_0|S_+)$$
Markov Models

- Set of states $Q = \{S_+, S_0, S_-\}$
- Transition probability matrix $A$
- Distribution on initial states $P_{init}$
- Set of end states $Q_{end}$

$P([S_0, S_0, S_+, S_+, S_0]) =

.33 * .3 * .4 * .3 * .6 = .0071$
Markov Models

(define transition-dists
  '(((interesting . (.3 .6 .1))
    (so-so . (.4 .3 .3))
    (boring . (.2 .6 .3))))

(define (transition state)
  (multinomial 'interesting so-so boring)
  (rest (assoc state transition-dists))))

(define (mm state n)
  (if (= n 0)
      '()
      (pair state
        (mm (transition state) (- n 1)))))
Hidden Markov Models

- Set of states $Q = \{E, \neg E\}$
- Transition probability matrix $A$
- Distribution on initial states $P_{\text{init}}$
- Set of end states $Q_{\text{end}}$

- Set of possible observations $V = \{S_+, S_0, S_-\}$
- Observation probability matrix $O$
Hidden Markov Models

- Set of states \( Q = \{E, \neg E\} \)
- Transition probability matrix \( A \)
- Distribution on initial states \( P_{init} \)
- Set of end states \( Q_{end} \)

Set of possible observations \( V = \{S_+, S_0, S_-\} \)
Observation probability matrix \( O \)

Markov assumption: \( P(q_i|q_1, \ldots, q_{i-1}) = P(q_i|q_{i-1}) \)

Output independence: \( P(o_i|q_1, \ldots, q_i, \ldots, q_T) = P(o_i|q_i) \)
Hidden Markov Models

(begin (mm state n)
  (if (= n 0)
    '()
    (pair state
      (mm (transition state) (- n 1))))))
(define (mm state n)
  (if (= n 0)
      '()
      (pair state
            (mm (transition state) (- n 1)))))

(define (hmm state n)
  (if (= n 0)
      '()
      (pair (observe state)
            (mm (transition state) (- n 1)))))
Hidden Markov Models

(define (mm state n)
  (if (= n 0)
    '()
    (pair state
      (mm (transition state) (- n 1)))))

(define (hmm state n)
  (if (= n 0)
    '()
    (pair (observe state)
      (mm (transition state) (- n 1)))))
Hidden Markov Models

- Set of states $Q = \{E, \neg E\}$
- Transition probability matrix $A$
- Distribution on initial states $P_{init}$
- Set of end states $Q_{end}$

- Set of possible observations $V = \{S_+, S_0, S_-\}$
- Observation probability matrix $O$
Inference by Enumeration
Inference by Enumeration

\[ P(o_{1:n}|q_{1:n}) = P(o_n|q_n) \cdots P(o_1|q_1) \]
Inference by Enumeration

\[ P(q_1:n) = P(q_n|q_{n-1}) \ldots P(q_2|q_1) P_{\text{init}}(q_1) \]
Inference by Enumeration

\[
P(o_1:n|q_1:n) = P(o_n|q_n) \ldots P(o_1|q_1)
\]

\[
P(q_1:n) = P(q_n|q_{n-1}) \ldots P(q_2|q_1)P_{init}(q_1)
\]

\[
P(o_1:n) = \sum_{q_1:n \in Q^n} P(o_1:n|q_1:n)P(q_1:n)
\]
Inference by Enumeration

\[
P(o_{1:n} | q_{1:n}) = P(o_n | q_n) \ldots P(o_1 | q_1)
\]

\[
P(q_{1:n}) = P(q_n | q_{n-1}) \ldots P(q_2 | q_1)P_{init}(q_1)
\]

\[
P(o_{1:n}) = \sum_{q_{1:n} \in Q^n} P(o_{1:n} | q_{1:n})P(q_{1:n})
\]
Inference by Enumeration

\[
P(o_{1:n}|q_{1:n}) = P(o_n|q_n) \ldots P(o_1|q_1)
\]

\[
P(q_{1:n}) = P(q_n|q_{n-1}) \ldots P(q_2|q_1)P_{init}(q_1)
\]

\[
P(o_{1:n}) = \sum_{q_{1:n} \in Q^n} P(o_{1:n}|q_{1:n})P(q_{1:n})
\]
Inference by Enumeration

\[
P(o_{1:n}|q_{1:n}) = P(o_n|q_n) \ldots P(o_1|q_1)
\]

\[
P(q_{1:n}) = P(q_n|q_{n-1}) \ldots P(q_2|q_1)P_{init}(q_1)
\]

\[
P(o_{1:n}) = \sum_{q_{1:n} \in Q^n} P(o_{1:n}|q_{1:n})P(q_{1:n})
\]
Inference by Enumeration

\[ P(o_{1:n}|q_{1:n}) = P(o_n|q_n) \ldots P(o_1|q_1) \]

\[ P(q_{1:n}) = P(q_n|q_{n-1}) \ldots P(q_2|q_1)P_{init}(q_1) \]

\[ P(o_{1:n}) = \sum_{q_{1:n} \in Q^n} P(o_{1:n}|q_{1:n})P(q_{1:n}) \]
Dynamic Programming

• Idea
  • Repeatedly reduce problem to smaller problems
  • Cache and reuse solutions to subproblems

• Requirements
  • Optimal substructure: Optimal solution to problem contains optimal solution to subproblems
  • Overlapping subproblems: Within solution to overall problem, need to solve subproblems multiple times

(Cormen, 2001)
Forward Algorithm

\[ P(o_{1:i}, q) = \begin{cases} 
  P_{\text{init}}(q)P(o_1|q) & \text{if } i = 1 \\
  \sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise}
\end{cases} \]

\[ P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q) \]
Forward Algorithm

\[
P(o_{1:i}, q) = \begin{cases} 
P_{init}(q)P(o_1|q) & \text{if } i = 1 \\ 
\sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise}
\end{cases}
\]

\[
P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)
\]

\[O(N|Q|^2)\]
**Forward Algorithm**

\[
P(o_{1:i}, q) = \begin{cases} 
P_{init}(q)P(o_1|q) & \text{if } i = 1 \\ 
\sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} 
\end{cases}
\]

\[
P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)
\]
Forward Algorithm

\[
P(o_{1:i}, q) = \begin{cases} 
  P_{\text{init}}(q) P(o_1|q) & \text{if } i = 1 \\
  \sum_r P(o_{1:(i-1)}, r) P(q|r) P(o_i|q) & \text{otherwise}
\end{cases}
\]

\[
P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q)
\]

\[O(N|Q|^2)\]
Forward Algorithm

\[ P(o_{1:i}, q) = \begin{cases} P_{\text{init}}(q)P(o_1 | q) & \text{if } i = 1 \\ \sum_{r \in Q} P(o_{1:(i-1)}, r) P(q | r) P(o_i | q) & \text{otherwise} \end{cases} \]

\[ P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q) \]
Forward Algorithm

\[ P(o_{1:i}, q) = \begin{cases} P_{\text{init}}(q)P(o_1|q) & \text{if } i = 1 \\ \sum_{r \in Q} P(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} \end{cases} \]

\[ P(o_{1:i}) = \sum_{q \in Q} P(o_{1:i}, q) \]
Viterbi Algorithm

\[
P_{\text{max}}(o_{1:i}, q) = \begin{cases} 
P_{\text{init}}(q)P(o_1|q) & \text{if } i = 1 \\ 
\max_{r \in Q} P_{\text{max}}(o_{1:(i-1)}, r)P(q|r)P(o_i|q) & \text{otherwise} 
\end{cases}
\]

\[
P_{\text{max}}(o_{1:i}) = \max_{q \in Q} P_{\text{max}}(o_{1:i}, q)
\]

\[O(N|Q|^2)\]
Dynamic Programming Algorithms

- Many DP algorithms exist ...
  - HMMs: Forward, Viterbi, Forward-Backward
  - PCFGs: Earley, CYK
  - MDPs: Bellman backups

- ... but probabilistic programming allows us to write many more models without known solutions

- Goal: Universal DP to make inference in some of these models tractable
Original program

```scheme
(define (game player)
  (if (flip .6)
      (not (game (not player)))
      (if player
          (flip .2)
          (flip .7))))
(game true)
```

Transformed program

```scheme
...((vector-ref op265 0) op265
  (vector
    (lambda (self266 tmp151)
      (let ((b135 tmp151))
        (let ((op267
              (vector-ref self266 2)))
          ((vector-ref op267 0) op267
            (vector
              (lambda (self268 tmp153)
                (make-application tmp153
                  (vector-ref self268 2))
                  #t))
              #t)
              (vector-ref self266 3)
              (vector-ref self266 4))))
            '61)
            (vector-ref self258 3)
            k134 game)
            game abs136)
    ...)
```

Factored computation graph

Subproblem dependency graph

Subproblem component DAG

Systems of polynomial equations

```
C1 = 1.0 * C
C:false = A:false + C1
C:true = A:true + C1
A1 = 1.0 * A
A2 = 0.4 * A1
A3 = 0.6 * A1
B1 = 1.0 * B
B2 = 0.6 * B1
B3 = 0.4 * B1
B:false = 0.3 * B3 + A:true * B2
B:true = 0.7 * B3 + A:false * B2
```

Marginal probabilities

```
<table>
<thead>
<tr>
<th></th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>0.8</td>
<td>0.2</td>
</tr>
<tr>
<td>A2</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>A3</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>B1</td>
<td>0.8</td>
<td>0.2</td>
</tr>
<tr>
<td>B2</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>B3</td>
<td>0.7</td>
<td>0.3</td>
</tr>
</tbody>
</table>
```

(Stuhlmüller, Roy, O’Donnell, and Goodman, under review)
DP for probabilistic programs

- Enumeration

  ```scheme
  (define (foo)
      (and (flip .2)
           (flip .3)))
  (foo)
  ```

- Factorization

- State merging

- Challenge: Recursive non-tail dependencies
DP for probabilistic programs

- Enumeration
- Factorization

\[
\text{(define (outer)}
\text{(if (flip .4)}
\text{(inner)}
\text{(not (inner))))}
\]

\[
\text{(define (inner)}
\text{(and (flip .2)}
\text{(flip .3))))}
\]

- State merging
- Challenge: Recursive non-tail dependencies
DP for probabilistic programs

- **Enumeration**

- **Factorization**

```scheme
(define (outer)
  (if (flip .4)
      (inner)
      (not (inner))))

(define (inner)
  (and (flip .2)
       (flip .3)))

(outer)
```

- **State merging**

- **Challenge: Recursive non-tail dependencies**
DP for probabilistic programs

• Enumeration
• Factorization

(define (outer)
  (if (flip .4)
      (inner)
      (not (inner))))

(define (inner)
  (and (flip .2)
       (flip .3)))

(outer)

• State merging

• Challenge: Recursive non-tail dependencies
DP for probabilistic programs

• Enumeration
• Factorization
• State merging

(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))
(tail)

• Challenge: Recursive non-tail dependencies
DP for probabilistic programs

- Enumeration
- Factorization
- State merging

(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))

(tail)

- Challenge: Recursive non-tail dependencies
DP for probabilistic programs

- Enumeration
- Factorization
- State merging

(\texttt{define (tail)}
  (\texttt{if (flip .4)}
    (\texttt{tail)}
    (flip .3)))

(tail)

- Challenge: Recursive non-tail dependencies
DP for probabilistic programs

• Enumeration
• Factorization
• State merging

(define (tail)
  (if (flip .4)
      (tail)
      (flip .3)))
(tail)

• Challenge: Recursive non-tail dependencies
DP for probabilistic programs

- Enumeration
- Factorization
- State merging

(define (tail)
  (if (flip .4)
      (tail)
      (flip .3))
(tail)

- Challenge: Recursive non-tail dependencies
DP for probabilistic programs

- Enumeration
- Factorization
- State merging

\[(\text{define (tail)}
\quad \text{(if (flip .4)}
\quad \quad \text{(tail)}
\quad \quad \text{(flip .3)})\))
\]
\[(\text{tail})\]

\[
A = 1 + .4 \times A \\
B = .6 \times A \\
\text{True} = .3 \times B \\
\text{False} = .7 \times B
\]
Challenge: Recursive dependencies

```
(define (stack)
  (if (flip .4)
      (not (stack))
      (flip .3)))

(stack)
```
Challenge: Recursive dependencies

(define (stack)
  (if (flip .4)
      (not (stack))
      (flip .3))))

(stack)
Challenge: Recursive dependencies

(define (stack)
  (if (flip .4)
      (not (stack))
      (flip .3)))

(stack)
Challenge: Recursive dependencies

\[
(\text{define} \ (\text{stack}) \\
\quad (\text{if} \ (\text{flip} \ .4) \\
\quad \quad (\text{not} \ (\text{stack})) \\
\quad \quad (\text{flip} \ .3)))
\]

\[
(\text{stack})
\]

\[
\begin{align*}
A &= 1 \\
B &= .4 \times A \\
C &= .6 \times A
\end{align*}
\]

\[
(\text{stack}) : \text{True} = (\text{stack}) : \text{False} \times B + .3 \times C
\]

\[
(\text{stack}) : \text{False} = (\text{stack}) : \text{True} \times B + .7 \times C
\]
The image shows a diagram illustrating the transformation of a program into a factored computation graph. The original program is provided, along with diagrams of subproblem dependency graphs, factored computation graphs, and systems of polynomial equations. The transformed program and the factored computation graph are shown, along with marginal probabilities and systems of polynomial equations.
Original program

(define (game player)
  (if (flip .6)
      (not (game (not player)))
      (if player
          (flip .2)
          (flip .7))))

(game true)

Transformed program

... 

((vector-ref op265 0) op265
  (vector
    (lambda (self266 tmp151)
      (let ((b135 tmp151))
        (let ((op267
            (vector-ref self266 2)))
          ((vector-ref op267 0) op267
            (vector
              (lambda
                (self268 tmp153)
                (make-application tmp153
                  (vector-ref self268 2)
                  #t))
             '60
                (vector-ref self268 3))
              (vector-ref self266 4)))))
       '61 (vector-ref self258 3)
       k134 game)
       game abs136)
...

Subproblem dependency graph

Systems of polynomial equations

C1 = 1.0 * C

C2 = 1.0 * C

C3 = 1.0 * C
Original program

(define (game player)
  (if (flip .6)
      (not (game (not player)))
      (if player
          (flip .2)
          (flip .7))))

Transposed program

...((vector-ref op265 0) op265
  (vector
    (lambda (self266 tmp151)
      (let ((b135 tmp151))
        (let ((op267
          (vector-ref
            self266 2)))
          ((vector-ref op267 0)
            op267
            (vector
              (lambda
                (self268 tmp153)
                (make-application
                  tmp153
                  (vector-ref
                    self268 2)
                  #t))
              '60
              (vector-ref
                self266 3))
              (vector-ref self266
                4))))))
  '61 (vector-ref self258 3)
  kl34 game)
  game abs136)
...

Factored computation graph

Subproblem dependency graph

Systems of polynomial equations

\[ C1 = 1.0 \times C \]

\[ A1 = 1.0 \times A \]
(define (game player)
  (if (flip .6)
      (not (game (not player)))
      (if player
          (flip .2)
          (flip .7))))
(game true)

0.8

A:

1.0

B:

1.0

C:

1.0

C1:

A: false
false

A: true
true

A1:

0.6

0.4

A2:

B: false
false

0.2

0.8

A3:

B: true
false

B1:

A: false
false

B: true
true

B2:

0.3

0.7

B3:

A: false
true

A1 = 1.0 * A
(define (game player)
  (if (flip .6)
      (not (game (not player)))
      (if player
          (flip .2)
          (flip .7))))

(game true)

True
2.375E-01
False
0.7625

A
B
C

C\text{false} = A\text{false} \times C_1
C\text{true} = A\text{true} \times C_1

\begin{align*}
C_1 &= 1.0 \times C \\
C\text{false} &= A\text{false} \times C_1 \\
C\text{true} &= A\text{true} \times C_1
\end{align*}
\[
\begin{align*}
C_1 &= 1.0 \times C \\
C: false &= A: false \times C_1 \\
C: true &= A: true \times C_1 \\
A_1 &= 1.0 \times A \\
A_2 &= 0.4 \times A_1 \\
A_3 &= 0.6 \times A_1 \\
A: false &= 0.8 \times A_2 + B: true \times A_3 \\
A: true &= 0.2 \times A_2 + B: false \times A_3 \\
B_1 &= 1.0 \times B \\
B_2 &= 0.6 \times B_1 \\
B_3 &= 0.4 \times B_1 \\
B: false &= 0.3 \times B_3 + A: true \times B_2 \\
B: true &= 0.7 \times B_3 + A: false \times B_2
\end{align*}
\]
(define (game player)
  (if (flip .6)
      (not (game (not player)))
      (if player
          (flip .2)
          (flip .7))))
(game true)
Original program

```
(define (game player)
  (if (flip .6)
      (not (game (not player)))
      (if player
          (flip .2)
          (flip .7))))

(game true)
```

Transformed program

```
... ...
(vec-ref op265 0) op265
(vecor
  (lambda (self266 tmp151)
    (let ((b135 tmp151))
      (let ((op267
        (vecor-ref self266 2)))
        ((vecor-ref op267 0))
          op267
            (vecor
              (lambda
                (self266 tmp153)
                  (make-application
                    tmp153
                      (vecor-ref
                        self266 2))))))
    '60
    (vecor-ref self266 3))
    (vecor-ref self266 4)))
'61 (vecor-ref self258 3)
  (k134 game)
  game abs136)
... ...
```

Factored computation graph

Subproblem dependency graph

Subproblem component DAG

Systems of polynomial equations

```
C1 = 1.0 * C
C1 = 1.0 * C
A2 = 0.4 * A1
A3 = 0.6 * A1
A1 = 1.0 * A
A1 = 1.0 * A
A2 = 0.4 * A1
A3 = 0.6 * A1
A1 = 1.0 * A
A2 = 0.4 * A1
A3 = 0.6 * A1
B1 = 1.0 * B
B2 = 0.6 * B1
B3 = 0.4 * B1
B1 = 1.0 * B
B2 = 0.6 * B1
B3 = 0.4 * B1
B1 = 1.0 * B
B2 = 0.6 * B1
B3 = 0.4 * B1
```

Marginal probabilities
A simple coordination game

![Graph showing time vs. depth of recursion with two lines representing UDP and Rejection]
A simple coordination game

Recursion depth 0

Recursion depth 1

Recursion depth 2

Recursion depth 3

Time (s)

Depth of recursion

0 1 2 3 4 5

0 1 2 3 4 5

0 1 2 3 4 5

0 1 2 3 4 5

MCMC (KL divergence)

UDP

Rejection
Cosh is a Church implementation based on the UDP algorithm: github.com/stuhlmueller/cosh