Probabilistic Programming: From Principled Foundations, Through Efficient Implementation, To Innovative Applications

Jeffrey Mark Siskind
Purdue University

Thursday 10 July 2014

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-10-2-0060. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.
Outline

1. Principled Foundations
2. Efficient Implementation
3. Innovative Applications
4. The Blessing and The Curse of Probabilistic Programming
Outline

1. Principled Foundations
2. Efficient Implementation
3. Innovative Applications
4. The Blessing and The Curse of Probabilistic Programming
Probabilistic Scheme
Probabilistic Scheme ≡ Scheme

Probabilistic Scheme ≡ Scheme + Bernoulli Trials

**Probabilistic Scheme** $\equiv$ **Scheme** + **Bernoulli Trials** + **Partition Function**


(define (fold-distribution-thunk f i thunk)
  (call-with-current-continuation
   (lambda (c)
     (let ((accumulation i)
           (p 1)
           (saved-flip *flip*)
           (saved-bottom *bottom*)
           (saved-current-probability *current-probability*))
       (set! *flip*
         (lambda (alpha)
           (unless (<= 0 alpha 1) (error #f "Alpha not probability")))
           (cond ((zero? alpha) #f)
                 ((= alpha 1) #t)
                 (else
                  (call-with-current-continuation
                   (lambda (c)
                     (let ((saved-p p) (saved-bottom *bottom*))
                       (set! p (* alpha p))
                       (set! *bottom*
                         (lambda ()
                           (set! p (* (- 1 alpha) saved-p))
                           (set! *bottom* saved-bottom)
                           (c #f))
                         #t))))))))
       (set! *bottom*
         (lambda ()
           (set! *flip* saved-flip)
           (set! *bottom* saved-bottom)
           (set! *current-probability* saved-current-probability)
           (c accumulation)))
       (set! *current-probability* (lambda () p))
       (let ((value (thunk)))
         (set! accumulation (f value p accumulation))
         (bottom))))))

(define-syntax fold-distribution
  (syntax-rules ()
    ((fold-distribution f i e) (fold-distribution-thunk f i (lambda () e))))
(define-syntax distribution ...)

(define-syntax support ...)

(define-syntax probability
  (syntax-rules ()
    ((probability e)
      (fold-distribution
        (lambda (value p accumulation) (if value (+ p accumulation) accumulation)) 0 e)))))

(define-syntax expected-value ...)

(define-syntax entropy ...)

(define-syntax most-likely-value-in-bag ...)

(define-syntax probability-of-most-likely-value-in-bag
  (syntax-rules ()
    ((probability-of-most-likely-value-in-bag e)
      (fold-distribution
        (lambda (value p accumulation) (max p accumulation)) 0 e)))))
(define-syntax distribution ...)

(define-syntax support ...)

(define-syntax probability
  (syntax-rules ()
    ((probability e)
      (fold-distribution
       (lambda (value p accumulation) (if value (+ p accumulation) accumulation))
       0
       e))))

(define-syntax expected-value ...)

(define-syntax entropy ...)

(define-syntax most-likely-value-in-bag ...)

(define-syntax probability-of-most-likely-value-in-bag
  (syntax-rules ()
    ((probability-of-most-likely-value-in-bag e)
      (fold-distribution
       (lambda (value p accumulation) (max p accumulation)) 0 e)))))

217 lines in R6RS
Discrete Graphical Model Scheme
**Discrete Graphical Model Scheme ≡ Probabilistic Scheme**
Discrete Graphical Model Scheme $\equiv$ Probabilistic Scheme + Arc Consistency


(define-record-type distribution-variable
  (fields (mutable distribution) (mutable demons)))

(define (assert-constraint-ac! constraint ds)
  (for-each
    (lambda (d)
      (attach-demon!
        (lambda ()
          (for-each-indexed
            (lambda (d i)
              (restrict-distribution!
                d
                (the-elements
                  (lambda (x)
                    (let loop ((ds ds) (xs '()) (j 0))
                      (if (null? ds)
                          (apply constraint (reverse xs))
                          (if (= j i)
                              (loop (rest ds) (cons x xs) (+ j 1))
                              (some-element
                                (lambda (x) (loop (rest ds) (cons x xs) (+ j 1)))
                                (first ds))))))
                d))
            ds))
        d))
    ds))

(define (stochastic-solution ds)
  (let loop ((ds ds) (xs '()))
    (if (null? ds)
      (reverse xs)
      (let ((pair
              (draw-pair (distribution-variable-distribution (first ds)))))
        (restrict-distribution! (first ds) (list pair))
        (loop (rest ds) (cons (first pair) xs))))))
(define (stochastic-branch-and-bound-solution ds)
  (let ((best 0))
    (let loop ((ds ds) (xs '()))
      (when (<= (fold (lambda (v d)
                       (* v
                          (map-reduce
                            max 0 cdr (distribution-variable-distribution d))))
                  (current-probability)
                  ds)
          best)
        (bottom))
      (cond ((null? ds)
              (set! best (current-probability))
                (reverse xs))
            (else
              (let ((pair
                       (draw-pair (distribution-variable-distribution (first ds))))))
              (restrict-distribution! (first ds) (list pair))
              (loop (rest ds) (cons (first pair) xs))))))))
(define (stochastic-branch-and-bound-solution ds)
  (let ((best 0))
    (let loop ((ds ds) (xs '()))
      (when (<= (fold (lambda (v d)
                        (* v
                           (map-reduce
                            max 0 cdr (distribution-variable-distribution d))))
                     (current-probability)
                     ds)
             best)
        (bottom))
      (cond ((null? ds)
                    (set! best (current-probability))
                    (reverse xs))
            (else
             (let ((pair
                        (draw-pair (distribution-variable-distribution (first ds))))
                      (restrict-distribution! (first ds) (list pair))
                      (loop (rest ds) (cons (first pair) xs))))))))

206 lines in R6RS
Schwish ≡ Discrete Graphical Model Scheme
Schwish $\equiv$ Discrete Graphical Model Scheme + WISH

(define (assert-random-xor-constraint! ds)
  (let ((b (random-boolean)))
    (apply assert-stochastic-constraint!
      (lambda vs (eq? (fold xor #f vs) b))
      (random-subset ds))))

(define (wish-find-constrained-max ds i)
  (most-likely-probability
   (begin (for-each-n (lambda (j) (assert-random-xor-constraint! ds)) i)
          (stochastic-branch-and-bound-solution ds))))

(define (wish ds delta alpha)
  (let* ((m (length ds))
         (xs (map-n (lambda (i) (median (map-n (lambda (t) ...
               (/ (log (/ m delta)) alpha)))))) (+ m 1)))
    (+ (first xs) (summation (lambda (x i) (* x (expt 2.0 i))) (rest xs)))))
(define (assert-random-xor-constraint! ds)
  (let ((b (random-boolean)))
    (apply assert-stochastic-constraint!
      (lambda vs (eq? (fold xor #f vs) b))
      (random-subset ds))))

(define (wish-find-constrained-max ds i)
  (most-likely-probability
   (begin (for-each-n (lambda (j) (assert-random-xor-constraint! ds)) i)
     (stochastic-branch-and-bound-solution ds))))

(define (wish ds delta alpha)
  (let* ((m (length ds))
         (xs (map-n (lambda (i)
            (median
             (map-n (lambda (t) (wish-find-constrained-max ds i))
              (inexact->exact (ceiling (/ (log (/ m delta)) alpha))))))))
         (+ m 1)))
    (+ (first xs) (summation (lambda (x i) (* x (expt 2.0 i))) (rest xs))))

49 lines in R6RS
\texttt{VLAD} $\equiv$ \texttt{SCHEME}
vlad \equiv \text{Scheme} + \nabla


VLAD $\equiv$ SCHEME $+ \nabla + \text{Gradient based Optimization}$


VLAD ≡ SCHEME + ∇ + Gradient based Optimization

arg max \( \Pr(x | \theta) \)


VLAD \equiv \text{SCHEME} + \nabla + \text{Gradient based Optimization}

\text{arg max}_{\theta} \Pr(x|\theta)


641 lines in R6RS
Outline

1. Principled Foundations
2. Efficient Implementation
3. Innovative Applications
4. The Blessing and The Curse of Probabilistic Programming
STALIN∇
\[ \text{STALIN} \triangledown \equiv \text{STALIN} \]

\text{STALIN} \triangledown \equiv \text{STALIN} + \text{Transformation based AD}


\[ \text{STALIN} \nabla \equiv \text{STALIN} + \boxed{\text{Transformation based AD}} + \text{CF}(\infty) \]


STALIN∇ ≡ STALIN + Transformation based AD + CF(∞)


26,384 lines in R4RS
implement interpreter for PROBABILISTIC SCHEME in VLAD
implement interpreter for PROBABILISTIC SCHEME in VLAD
parameter estimation via gradient-based optimization of likelihood
- implement interpreter for PROBABILISTIC SCHEME in VLAD
- parameter estimation via gradient-based optimization of likelihood
- take gradient through interpreter
implement interpreter for PROBABILISTIC SCHEME in VLAD
parameter estimation via gradient-based optimization of likelihood
take gradient through interpreter
compile with STALIN∇
\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]
\( P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \)

\[ \Pr(x_0 \mapsto \text{true}) = p_0 \quad \Pr(x_0 \mapsto \text{false}) = 1 - p_0 \]
\[ \Pr(x_1 \mapsto \text{true}) = p_1 \quad \Pr(x_1 \mapsto \text{false}) = 1 - p_1 \]
\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]

\[
\begin{align*}
\Pr(x_0 \mapsto \text{true}) &= p_0 & \Pr(x_0 \mapsto \text{false}) &= 1 - p_0 \\
\Pr(x_1 \mapsto \text{true}) &= p_1 & \Pr(x_1 \mapsto \text{false}) &= 1 - p_1 \\
\Pr(\mathcal{E}(P) = 0 | p_0, p_1) &= p_0 \\
\Pr(\mathcal{E}(P) = 1 | p_0, p_1) &= (1 - p_0)p_1 \\
\Pr(\mathcal{E}(P) = 2 | p_0, p_1) &= (1 - p_0)(1 - p_1)
\end{align*}
\]
\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]

\[
\begin{align*}
\Pr(x_0 \mapsto \text{true}) &= p_0 & \Pr(x_0 \mapsto \text{false}) &= 1 - p_0 \\
\Pr(x_1 \mapsto \text{true}) &= p_1 & \Pr(x_1 \mapsto \text{false}) &= 1 - p_1
\end{align*}
\]

\[
\begin{align*}
\Pr(\mathcal{E}(P) = 0|p_0, p_1) &= p_0 \\
\Pr(\mathcal{E}(P) = 1|p_0, p_1) &= (1 - p_0)p_1 \\
\Pr(\mathcal{E}(P) = 2|p_0, p_1) &= (1 - p_0)(1 - p_1)
\end{align*}
\]

\[
\prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v|p_0, p_1) = p_0(1 - p_0)^3p_1(1 - p_1)^2
\]
\( P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \)

\[
\begin{align*}
\Pr(x_0 \mapsto \text{true}) &= p_0 & \Pr(x_0 \mapsto \text{false}) &= 1 - p_0 \\
\Pr(x_1 \mapsto \text{true}) &= p_1 & \Pr(x_1 \mapsto \text{false}) &= 1 - p_1
\end{align*}
\]

\[
\begin{align*}
\Pr(\mathcal{E}(P) = 0 | p_0, p_1) &= p_0 \\
\Pr(\mathcal{E}(P) = 1 | p_0, p_1) &= (1 - p_0)p_1 \\
\Pr(\mathcal{E}(P) = 2 | p_0, p_1) &= (1 - p_0)(1 - p_1)
\end{align*}
\]

\[
\prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = p_0(1 - p_0)^3 p_1(1 - p_1)^2
\]

\[
\arg \max_{p_0, p_1} \prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = \left\langle \frac{1}{4}, \frac{1}{3} \right\rangle
\]
(gradient-ascent
(lambda (p)
  (let ((tagged-distribution
         (evaluate (if x0 then 0 else if x1 then 1 else 2
                     (list Pr(x₀ ⟷ true) = p₀  Pr(x₀ ⟷ false) = 1 − p₀
                         Pr(x₁ ⟷ true) = p₁  Pr(x₁ ⟷ false) = 1 − p₁
                         ...) ))))

(map-reduce
  * 1.0
  (lambda (value)
    (likelihood value tagged-distribution))
  '((0 1 2 2))
  '((0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
 (lambda (p)
 (let ((tagged-distribution
     (evaluate
       (if x0 then 0 else if x1 then 1 else 2
       (list
         Pr(x0 -> true) = p0  Pr(x0 -> false) = 1 - p0
         Pr(x1 -> true) = p1  Pr(x1 -> false) = 1 - p1
         ...)))))
     (map-reduce
       * 1.0
       (lambda (value)
         (likelihood value tagged-distribution))
       '(0 1 2 2)))
   '(0.5 0.5)
   1000.0
   0.1))
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
       (evaluate
         (if x0 then 0 else if x1 then 1 else 2
           (list Pr(x0 \rightarrow true) = p0  Pr(x0 \rightarrow false) = 1 - p0
                     Pr(x1 \rightarrow true) = p1  Pr(x1 \rightarrow false) = 1 - p1
                     ...
         )))
     (map-reduce
      * 1.0
      (lambda (value)
        (likelihood value tagged-distribution))
      '(0 1 2 2)))
    '(0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
           (evaluate
            (if x
                0
                (if x
                    1
                    2))
           (list
            Pr(x0 \rightarrow \text{true}) = p_0 \quad Pr(x0 \rightarrow \text{false}) = 1 - p_0
            Pr(x1 \rightarrow \text{true}) = p_1 \quad Pr(x1 \rightarrow \text{false}) = 1 - p_1
            ...))))

   (map-reduce
    * 1.0
    (lambda (value)
      (likelihood value tagged-distribution))
    '(0 1 2 2)))))

'(0.5 0.5)
1000.0
0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
            (evaluate if x0 then 0 else if x1 then 1 else 2
                       (list Pr(x0 \to true) = p0 Pr(x0 \to false) = 1 - p0
                           Pr(x1 \to true) = p1 Pr(x1 \to false) = 1 - p1
                           ...)))))
     (map-reduce
      * 1.0
      (lambda (value)
        (likelihood value tagged-distribution))
      '(0 1 2 2)))
    '(0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
     (evaluate if x0 then 0 else if x1 then 1 else 2
       (list Pr(x0 \rightarrow true) = p0  Pr(x0 \rightarrow false) = 1 - p0
            Pr(x1 \rightarrow true) = p1  Pr(x1 \rightarrow false) = 1 - p1
            ...))))

   (map-reduce
    * 1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    `(0 1 2 2))))

`(0.5 0.5)
1000.0
0.1)
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
       (evaluate if x0 then 0 else if x1 then 1 else 2
       (list Pr(x0 ↦ true) = p0  Pr(x0 ↦ false) = 1 - p0
             Pr(x1 ↦ true) = p1  Pr(x1 ↦ false) = 1 - p1
             ...)))

       (map-reduce
        * 1.0
        (lambda (value)
         (likelihood value tagged-distribution))
        '(0 1 2 2)))

       '(0.5 0.5)
       1000.0
       0.1))

Siskind (Purdue)
PPAML PI Meeting
10 July 2014
17 / 31
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
     (evaluate if x0 then 0 else if x1 then 1 else 2
               (list Pr(x0 → true) = p0  Pr(x0 → false) = 1 − p0
                    Pr(x1 → true) = p1  Pr(x1 → false) = 1 − p1
                    ...)))))
    (map-reduce
      * 1.0
      (lambda (value)
        (likelihood value tagged-distribution))
      '(0 1 2 2)))
  '(0.5 0.5)
  1000.0
  0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
          (evaluate
            (if x0 then 0 else if x1 then 1 else 2
             (list
              Pr(x0 ↦ true) = p0  Pr(x0 ↦ false) = 1 – p0
              Pr(x1 ↦ true) = p1  Pr(x1 ↦ false) = 1 – p1
              ...))))
      (map-reduce
       * 1.0
       (lambda (value)
        (likelihood value tagged-distribution))
       '(0 1 2 2)))
    '(0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
      (evaluate
        (if x0 then 0 else if x1 then 1 else 2
          (list
            Pr(x0 → true) = p0  Pr(x0 → false) = 1 − p0
            Pr(x1 → true) = p1  Pr(x1 → false) = 1 − p1
            ...))))
      (map-reduce
        * 1.0 (lambda (value) (likelihood value tagged-distribution))
        ’(0 1 2 2))))
     ’(0.5 0.5)
  1000.0 0.1))
static void f2679(double a_f2679_0, double a_f2679_1, double a_f2679_2, double a_f2679_3) {
    int t272381 = ((a_f2679_2 == 0.) ? 0 : 1);
    double t272406;
    double t272405;
    double t272404;
    double t272403;
    double t272402;
    if ((t272381 == 0)) {
        double t272480 = (1. - a_f2679_0);
        double t272572 = (1. - a_f2679_1);
        double t273043 = (a_f2679_0 + 0.);
        double t274185 = (t272480 * a_f2679_1);
        double t274426 = (t274185 + 0.);
        double t275653 = (t272480 * t272572);
        double t275894 = (t275653 + 0.);
        double t277121 = (t272480 * t272572);
        double t277362 = (t277121 + 0.);
        double t277431 = (t277362 * 1.);
        double t277436 = (t275894 * t277431);
        double t277441 = (t274426 * t277436);
        double t277446 = (t273043 * t277441);
        ...
        double t1777107 = (t1774696 + t1715394);
        double t1777194 = (0. - t1745420);
        double t1778533 = (t1777194 + t1419700);
        t272406 = a_f2679_0;
        t272405 = a_f2679_1;
        t272404 = t277446;
        t272403 = t1778533;
        t272402 = t1777107;}
    else {(...)
        r_f2679_0 = t272406;
        r_f2679_1 = t272405;
        r_f2679_2 = t272404;
        r_f2679_3 = t272403;
        r_f2679_4 = t272402;}
}
<table>
<thead>
<tr>
<th>Scheme</th>
<th>probabilistic-Scheme</th>
<th>probabilistic-Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLAD</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>STALINGRAD</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>ML</td>
<td>106.45 124.95</td>
<td>789.41 483.47</td>
</tr>
<tr>
<td>MLTON</td>
<td>215.73 538.68</td>
<td>1207.13 1534.61</td>
</tr>
<tr>
<td>OCAML</td>
<td>197.75 272.45</td>
<td>2448.02 1471.94</td>
</tr>
<tr>
<td>SML/NJ</td>
<td>832.92 1048.11</td>
<td>14422.16 8286.06</td>
</tr>
<tr>
<td>Haskell</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GHC</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Scheme</td>
<td>832.92 1048.11</td>
<td></td>
</tr>
<tr>
<td>Bigloo</td>
<td>2305.98 3283.00</td>
<td>66948.70 37792.84</td>
</tr>
<tr>
<td>Chicken</td>
<td>879.88 1153.86</td>
<td>24316.03 13649.81</td>
</tr>
<tr>
<td>Gambit</td>
<td>437.46 531.10</td>
<td>8242.92 4845.86</td>
</tr>
<tr>
<td>Ikarus</td>
<td>1651.01 1673.22</td>
<td>25589.62 14833.53</td>
</tr>
<tr>
<td>Larceny</td>
<td>3491.10 4130.19</td>
<td>85819.57 48335.38</td>
</tr>
<tr>
<td>MIT Scheme</td>
<td>5289.17 5929.14</td>
<td>154206.95 83480.27</td>
</tr>
<tr>
<td>MzC</td>
<td>6235.78 7134.71</td>
<td>166129.12 91630.70</td>
</tr>
<tr>
<td>MzScheme</td>
<td>682.15 794.31</td>
<td>10530.66 5980.27</td>
</tr>
<tr>
<td>Scheme-&gt;C</td>
<td>6456.99</td>
<td>80100.23</td>
</tr>
<tr>
<td>Scmutil</td>
<td>1240.73 1137.41</td>
<td>22511.79 10986.43</td>
</tr>
</tbody>
</table>

- not implemented but could implement, including FORTRAN, C, and C++
- not implemented in existing tool
- can’t implement
Outline

1. Principled Foundations
2. Efficient Implementation
3. Innovative Applications
4. The Blessing and The Curse of Probabilistic Programming
B: a video, represented as a set of detections in each frame, each detection annotated with features

\( \Lambda \): a lexicon, models for each word

\( S(B, s, \Lambda) = \Pr(J | B) \prod \Pr(B_j | s, \Lambda) \)
**B**: a video, represented as a set of detections in each frame, each detection annotated with features
**B**: a video, represented as a set of detections in each frame, each detection annotated with features

**s**: a sentence, represented as a sequence of words
**B**: a video, represented as a set of detections in each frame, each detection annotated with features

**s**: a sentence, represented as a sequence of words

**Λ**: a lexicon, models for each word
$\mathbf{B}$: a video, represented as a set of detections in each frame, each detection annotated with features

$s$: a sentence, represented as a sequence of words

$\Lambda$: a lexicon, models for each word

$$S(\mathbf{B}, s, \Lambda)$$
**Sentence Tracker**

**B:** a video, represented as a set of detections in each frame, each detection annotated with features

**s:** a sentence, represented as a sequence of words

**Λ:** a lexicon, models for each word

\[ S(B, s, \Lambda) = \mathbb{E}_{Pr(j|B)}[Pr(B_j|s, \Lambda)] \]
\( \mathbf{B} \): a video, represented as a set of detections in each frame, each detection annotated with features

\( \mathbf{s} \): a sentence, represented as a sequence of words

\( \Lambda \): a lexicon, models for each word

\[ S(\mathbf{B}, \mathbf{s}, \Lambda) = \mathbb{E}_{\Pr(\mathbf{J}|\mathbf{B})} [\Pr(\mathbf{B}_J|\mathbf{s}, \Lambda)] = \sum_{\mathbf{J}} \Pr(\mathbf{J}|\mathbf{B}) \Pr(\mathbf{B}_J|\mathbf{s}, \Lambda) \]
Given a video clip $B$ and a lexicon $\Lambda$, produce a sentential description $s^*$ of the video clip.
Using the Sentence Tracker for Generation

Given a video clip $\mathbf{B}$ and a lexicon $\Lambda$, produce a sentential description $s^*$ of the video clip.

$$s^* = \arg \max_s S(\mathbf{B}, s, \Lambda)$$
Given a video clip $B$ and a lexicon $\Lambda$, produce a sentential description $s^*$ of the video clip.

$$s^* = \arg \max_s S(B, s, \Lambda)$$

Given a set \( \{B_1, \ldots, B_M\} \) of video clips, a sentential query \( s \), and a lexicon \( \Lambda \), find the video clip \( B_{m^*} \) that best matches the query.
Given a set \( \{ \mathbf{B}_1, \ldots, \mathbf{B}_M \} \) of video clips, a sentential query \( s \), and a lexicon \( \Lambda \), find the video clip \( \mathbf{B}_{m^*} \) that best matches the query.

\[
m^* = \arg \max_m S(\mathbf{B}_m, s, \Lambda)
\]
Using the Sentence Tracker for Retrieval

Given a set \( \{B_1, \ldots, B_M\} \) of video clips, a sentential query \( s \), and a lexicon \( \Lambda \), find the video clip \( B_{m^*} \) that best matches the query.

\[
m^* = \arg \max_m S(B_m, s, \Lambda)
\]

Given a training set \( \{(B_1, s_1), \ldots, (B_M, s_M)\} \) of video clips paired with sentential descriptions, learn the best lexicon \( \Lambda^* \).

\[ \Lambda^* = \arg \max_{\Lambda} \prod_{m=1}^{M} S(B_m, s_m, \Lambda) \]

Given a training set \( \{(B_1, s_1), \ldots, (B_M, s_M)\} \) of video clips paired with sentential descriptions, learn the best lexicon \( \Lambda^* \).

\[
\Lambda^* = \arg \max_{\Lambda} \prod_{m=1}^{M} S(B_m, s_m, \Lambda)
\]
Using the Sentence Tracker for Acquisition

Given a training set \( \{(B_1, s_1), \ldots, (B_M, s_M)\} \) of video clips paired with sentential descriptions, learn the best lexicon \( \Lambda^* \).

\[
\Lambda^* = \arg \max_{\Lambda} \prod_{m=1}^{M} S(B_m, s_m, \Lambda)
\]

Live Demo
Live Demo
Live Demo
Live Demo
Outline

1. Principled Foundations
2. Efficient Implementation
3. Innovative Applications
4. The Blessing and The Curse of Probabilistic Programming
The Blessing
existing object detection method: SVM over HOG
(Dalal & Triggs, Felzenszwalb et al.)

existing action recognition method: time series modeled by HMM
(trained with Baum-Welch)

idea: Felzenszwalb + Baum-Welch

HMM with SVM output model

SVM output model over HOG and HOF (retinotopic flow fields)

train by gradient ascent over aggregate score of a labeled dataset

classify by selecting class with highest score on unseen video
existing object detection method: SVM over HOG (Dalal & Triggs, Felzenszwalb et al.)
existing object detection method: SVM over HOG
(Dalal & Triggs, Felzenszwalb et al.)

existing action recognition method: time series modeled by HMM
(trained with Baum-Welch)
existing object detection method: SVM over HOG (Dalal & Triggs, Felzenszwalb et al.)
existing action recognition method: time series modeled by HMM (trained with Baum-Welch)
idea: Felzenszwalb + Baum-Welch
existing object detection method: SVM over HOG
(Dalal & Triggs, Felzenszwalb et al.)

existing action recognition method: time series modeled by HMM
(trained with Baum-Welch)

idea: Felzenszwalb + Baum-Welch

HMM with SVM output model
- existing object detection method: SVM over HOG (Dalal & Triggs, Felzenszwalb et al.)
- existing action recognition method: time series modeled by HMM (trained with Baum-Welch)
- idea: Felzenszwalb + Baum-Welch
- HMM with SVM output model
- SVM output model over HOG and HOF (retinotopic flow fields)
existing object detection method: SVM over HOG (Dalal & Triggs, Felzenszwalb et al.)

existing action recognition method: time series modeled by HMM (trained with Baum-Welch)

idea: Felzenszwalb + Baum-Welch

HMM with SVM output model

SVM output model over HOG and HOF (retinotopic flow fields)

train by gradient ascent over aggregate score of a labeled dataset
existing object detection method: SVM over HOG
(Dalal & Triggs, Felzenszwalb et al.)
existing action recognition method: time series modeled by HMM
(trained with Baum-Welch)
idea: Felzenszwalb+Baum-Welch
HMM with SVM output model
SVM output model over HOG and HOF (retinotopic flow fields)
train by gradient ascent over aggregate score of a labeled dataset
classify by selecting class with highest score on unseen video
The Curse