Recognizing Multitasked Activities from Video using Stochastic Context-Free Grammar

(Darnell Moore and Irfan Essa)

[**AAAI 2002, CVPR 2001** (Workshop on Models vs. Exemplars)]

Presented by:

Asaad Hakeem

Objectives

- Recognizing multitasked activities from videos
- Stochastic Context Free Grammar (SCFG) for defining rules of event occurrences
- Earley-Stolcke parsing for detection of event sequence
- Detection & recovery of insertion, deletion and substitution errors
- High-level behavior assessment from event sequence

Related Work

- Earley, J. C. 1968. *An Efficient Context-Free Parsing Algorithm.* Ph.D. Dissertation, Carnegie-Mellon University.
- Stolcke, A. 1994. *Bayesian Learning of Probabilistic Language Models.* Ph.D., University of California at Berkeley.
- Ivanov, Y., and Bobick, A. 2000. Recognition of visual activities and interactions by stochastic parsing. *PAMI* 22(8):852–872.

Stochastic Context Free Grammar

- A context free grammar has a non-terminal producing a string including both terminals and/or non-terminals. E.g.

  S → A B c (non-terminals are capitalized, terminals are small-cased)
- SCFG is a grammar having probability associated with each rule
- This has added advantages:
  - Finding maximum probability sequence of events
  - Pruning parses that have low accumulated probability
- Probabilities are calculated from training data

Probability Representation

- Probability of each rule is given by:

$$P(X \to \lambda) = \frac{c(X \to \lambda)}{\sum_{\mu} c(X \to \mu)}$$

c (X→λ) = count of particular production
c (X→μ) = count of non-terminals produced by X

- Total probability of an event sequence is given by:

$$P_D(x) = \prod_{i=1}^{l} P_D(x_i)$$

$P_D(X_i)$ = probability of an event

- Probability is normalized for longer strings by:

$$\widetilde{P}_D(x) = \frac{1}{l} \sum_{i=1}^{l} P_D(x_i)$$

l = length of the string

Earley-Stolcke Parsing

- Earley introduced a CFG parsing mechanism that:
  - Does not repeat parsing of already parsed sub-trees
  - Has N+1 tables for N number of words (events)
  - State in a table has:
    - Sub-tree for a single grammar rule
    - Progress made in completing sub-tree
    - Position of sub-tree with respect to input
  - E.g.
    S → . A B [0,0] Prediction
    S → A . B [0,1] In-progress
    S → A B . [0,2] Completed
- Parsing occurs in three stages:
  - Prediction
  - Scanning
  - Completion

Prediction Stage

- Creates lists of all the states possible on prior input
- These states provide candidate terminal symbols at the next position in the string

Predict (A → α . B β [i, j])
    for each (B → γ) in Grammar rules
        Enqueue ((B → . γ [j, j] in table [j])
    end

α=previously parsed string, β=string to parse after B
i=current position, B = non-terminal
j=current table, B → γ = all the productions of B

Scanning Stage

- Next input symbol is matched with all states
- Ensures terminal symbols produced matches input string
- Promotes the states for the next iteration

Scan (A → α . B β [i, j])
    if (B produces and matches the current event)
        Enqueue ((B → γ . [j, j+1] [b, a] in table [j+1])
    end

α=previously parsed string, β=string to parse after B
i=current position, B = non-terminal
j=current table, B → γ = matching terminal production of B
a=forward probabilities b=inner probabilities

Completion Stage

- Completion stage updates the positions in all pending derivations in the confirmed scanned states
- Completion corresponds to an end of a non-terminal expansion initiated by the prediction stage
- A state is complete if:
  - Sub-sting $x_i \ldots x_l$ has been fully expanded, and
  - It is syntactically correct B → γ . [j, k]

Complete (B → γ . [j, k])
    for each ((A → α . B β [i, j]) in table [j])
        Enqueue ((A → α B . β [i, k] {append completer id }) in table [k])
    end

α=previously parsed string,
i=arbitrary position in j^{th} table,
j=current position,
k=current table,

β=string to parse after B
B = non-terminal
B → γ = completed production of B
{ id_1,… }= list of ids reaching this production

Parsing using Probabilities

- Stolcke added probabilities to each state
- There are two types of probabilities associated with each state:
  - Forward probability α
  - Inner probability γ
- Forward probability is the likelihood of selecting next state x_i along with all the previously selected states $x_1 \ldots x_{i-1}$

  $\alpha = P(x_1) + P(x_2) + \ldots P(x_i)$
- Inner probability is the likelihood of generating a substring from the given non-terminal $x_k \ldots x_{i-1}$

  $\gamma = P(x_k) + P(x_{k+1}) + \ldots P(x_i)$,

  given: B → λ . μ [k, i]

Parsing using Probabilities (contd.)

- The probability of producing symbol x_i was given by $P_D(x_i)$
- Forward and inner probabilities are updated during scanning by:

  $\alpha` = \alpha(X \rightarrow \lambda\ .\ a\ \mu\ [1, i]) * P_D(a)$

  $\gamma` = \gamma(X \rightarrow \lambda\ .\ a\ \mu\ [k, i]) * P_D(a)$

  where 'a' is the terminal in the input in state set (table) i
- Updated $\alpha`$ and $\gamma`$ reflect:
  - Weight of the likelihood of competing derivations
  - Certainty associated with the scanned input symbol

Selecting Maximum Likelihood Parse

- Viterbi method for parsing a string *x*, with most likely probability from all the derivations of *x*
- Each state holds the maximal path probability leading to it
- Viterbi probability v is propagated similar to γ, except at completion, where summation is replaced by maximum

$$v_i (X \rightarrow \lambda Y.\mu\ [k, i]) = \max_{\lambda,\mu}\{v_i (Y \rightarrow x.\ [j, i]) * v_j (X \rightarrow \lambda.Y\mu\ [k, j])\}$$

- By backtracking along the maximal predecessor states, the maximal probability parse will be recovered

$$X \rightarrow \lambda Y.\mu\ [k, i] = \operatorname{argmax}_{\lambda,\mu}\{v_i (Y \rightarrow x.\ [j, i]) * v_j (X \rightarrow \lambda.Y\mu\ [k, j])\}$$

Earley-Stolcke Algorithm

```
function Earley-Parse (events, grammar) returns tables
    Enqueue (γ → . S [0, 0] in table [0])   // dummy state for starting table
    for i=0 to length (events)
        for each state in table [i]
            if Incomplete (state) and Next-Cat (state) != Event
                Predict (state)
            else if Incomplete (state) and Next-Cat (state) = Event
                Scan (state)
            else
                Complete (state)
        end // for
    end // for
return (table)
end // function
```

Earley-Stolcke Algorithm (contd.)

```
function Generate-ParseTree (table) returns list //of successful parse trees
    for all entries in table [l] having starting productions
        trace back through ids in the parse list {}
        add probability of each production included in parse list
        divide the total probability with length of string l // likely probability
        EnqueueList (tree [i])          // enqueue tree in list of successful parses
    end // for
return (list)
end // function
```

- Find the maximum likely probability parse tree from the list of successful parse trees
- The events in the maximum likely parse tree are the most likely sequence of events

Sample Parse

- Consider a simple parse of string 'abc' given the following grammar:

S →AB	[1.0]	B→BC	[0.5]
A→aA	[0.5]	B→b	[0.5]
A→a	[0.5]	C→c	[1.0]

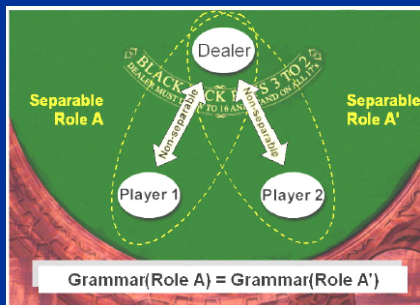
table[0]	[k, i]	[γ , α]		
S1: γ →.S	[0,0]	[1.0,1.0]	{}	Dummy start state
S2: S →.AB	[0,0]	[1.0,1.0]	{}	Predictor
S3: A→.aA	[0,0]	[1.0,0.5]	{}	Predictor
S4: A→.a	[0,0]	[0.5,0.5]	{}	Predictor
table[1]				
S5: A→a.	[0,1]	[0.5,0.5]	{}	Scanner
S6: S→A.B	[0,1]	[0.5,0.5]	{S5}	Completer
S7: B→.BC	[1,1]	[0.0,0.5]	{}	Predictor
S8: B→.b	[1,1]	[0.0,0.5]	{}	Predictor

Sample Parse (contd.)

table[2]				
S9: B→b.	[1,2]	[0.5,0.5]	{}	Scanner
S10: B→B.C	[1,2]	[0.5,0.5]	{S9}	Completer
S11: S→AB.	[0,2]	[0.75,0.75]	{S9,S10}	Completer
S12: C→.c	[1,2]	[0.0,0.5]	{}	Predictor
table[3]				
S13: C→c.	[2,3]	[0.5,0.67]	{}	Scanner
S14: B→BC.	[1,3]	[0.75,0.67]	{S9,S13}	Completer
S15: S→AB.	[0,3]	[0.67,0.67]	{S5,S14}	Completer

Parsing Separable Activities

- Activities with separable groups are independent interactive relationships between two or more agents
- In case of Blackjack, each player dealer pair is a separable group
- Development of SCFG that describe non-separable interactions



Seperable, Independent roles

SCFG for Blackjack

Production Rules			Description
S	→ *AB*	[1.0]	Blackjack → "play game" "determine winner"
A	→ *CD*	[1.0]	play game → "setup game" "implement strategy"
B	→ *EF*	[1.0]	determine winner → "eval. strategy" "cleanup"
C	→ *HI*	[1.0]	setup game → "place bets" "deal card pairs"
D	→ *GK*	[1.0]	implement strategy → "player strategy"
E	→ *LKM*	[0.6]	eval. strategy → "dealer down-card" "dealer hits" "player down-card"
	→ *LM*	[0.4]	eval. strategy → "dealer down-card" "player down-card"
F	→ *NO*	[0.5]	cleanup → "settle bet" "recover card"
	→ *ON*	[0.5]	→ "recover card" "settle bet"
G	→ *J*	[0.8]	player strategy → "Basic Strategy"
	→ *Hf*	[0.1]	→ "Splitting Pair"
	→ *bfffH*	[0.1]	→ "Doubling Down"
H	→ *l*	[0.5]	place bets
	→ *lH*	[0.5]	
I	→ *ffI*	[0.5]	deal card pairs
	→ *ee*	[0.5]	
J	→ *f*	[0.8]	Basic strategy
	→ *fJ*	[0.2]	
K	→ *e*	[0.6]	house hits
	→ *eK*	[0.4]	
L	→ *ae*	[1.0]	Dealer downcard
M	→ *dh*	[1.0]	Player downcard
N	→ *k*	[0.16]	settle bet
	→ *kN*	[0.16]	
	→ *j*	[0.16]	
	→ *jN*	[0.16]	
	→ *i*	[0.18]	
	→ *iN*	[0.18]	
O	→ *a*	[0.25]	recover card
	→ *aO*	[0.25]	
	→ *b*	[0.25]	
	→ *bO*	[0.25]	

Symbol	Domain-Specific Events (Terminals)
a	dealer removed card from house
b	dealer removed card from player
c	player removed card from house
d	player removed card from player
e	dealer added card to house
f	dealer dealt card to player
g	player added card to house
h	player added card to player
i	dealer removed chip
j	player removed chip
k	dealer pays player chip
l	player bets chip

Primitive Events

- Terminal symbols are based on primitive events
- A separate symbolic string is maintained for each person p_m
- A relation between an event and person is based on:
  - Person in contact with an article
  - The owner of the article
- First measure is determined by overlap between image regions of the hands and objects
- Second measure is based on the proximity zone z_m (manually)
- Tags are attached during scanning when the next state is added:

  $X \rightarrow \lambda a.\mu\ [\alpha, \gamma, p_j, o(z_m)]\ [k, i+1]$, where $o(z_m)$ returns ID p_j based on zone

Player Behavior (Skill)

- A subset of production rules can be used to assess behavior
- Production rule G suggests player strategy and skill
  - Let P_ς be the subset denoting productions of G
  - Let b_ς be production likelihood of P_ς
  - Initially b_ς is set to be uniform i.e. $b_\varsigma = b_x = (\beta_1, \beta_1, \ldots \beta_n)/n$
  - Likelihood of selected rules by an individual is denoted by $\hat{b}_\varsigma$
  - Mean Square Error (MSE) is given by:

$$err(b_\varsigma - \hat{b}_\varsigma) = \frac{1}{n}\sum_{i=1}^{n}(\beta_i - \hat{\beta}_i)^2$$

  - The likelihood of behavior given the model is calculated by pair-wise distance measure of MSE by:

$$P(\hat{b}_\varsigma \mid b_\varsigma) = 1 - \sqrt{err(b_\varsigma - \hat{b}_\varsigma)}$$

Error Detection & Recovery

- Errors in input can generate ungrammatical strings, causing parsing algorithm to fail
- Three types of error detection and recovery are provided:
  - *Substitution error* occurs when wrong terminal string is generated, as actual event is not detected to be most likely
  - *Insertion error* occurs when spurious terminal string is generated, as an event is incorrectly detected (false positive)
  - *Deletion error* occurs when an event is not detected
- Parsing errors occur in scanning stage when input symbol does not match terminals from prediction stage
- Prediction stage is modified to expand all productions Y until next terminal is predicted (matched at scanning), i.e.

$X \rightarrow \lambda.\mu\ [\alpha,\gamma]\ [k, i] \Rightarrow Y \rightarrow .\nu\ [\alpha`, \gamma`]\ [i, i] \Rightarrow Y \rightarrow .a\xi\ [\alpha`, \gamma`]\ [i, i]$

Insertion Error

- Simply ignore the scanned terminal
- Return the state of the parser to previous point prior to scanning
- Same pending expansions of prediction are maintained

Substitution Error

- Promote each pending prediction as if it was actually scanned
- New hypothetical terminal is created resulting in multiple paths
- Hypothetical path is terminated if failure occurs at next step
- Actual likelihood of the event (terminal $P_D(a)$) is recovered by,

$$\alpha` = \alpha(Y \rightarrow .a\xi\ [\alpha`, \gamma`]\ [i, i])*P_D(a)$$

$$\gamma` = \gamma(Y \rightarrow .a\xi\ [\alpha`, \gamma`]\ [i, i])*P_D(a)$$

Deletion Error

- Promote each pending prediction as if it was actually scanned
- New hypothetical terminal is created resulting in multiple paths
- Proceed to the completion stage and modify probabilities at scanning by,

$$\alpha` = \alpha(Y \rightarrow .a\xi\ [\alpha`, \gamma`]\ [i, i])*\tilde{P}_D(a)$$

$$\gamma` = \gamma(Y \rightarrow .a\xi\ [\alpha`, \gamma`]\ [i, i])*\tilde{P}_D(a)$$

- At the next prediction stage there is no detection of likelihood for recovery
- At second scanning stage, the probabilities are recovered from original scan likelihood by,

$$\alpha` = \alpha(Y \rightarrow .a\xi\ [\alpha`, \gamma`]\ [i+1, i+1])*P_D(b)$$

$$\gamma` = \gamma(Y \rightarrow .a\xi\ [\alpha`, \gamma`]\ [i+1, i+1])*P_D(b)$$

- This method guarantees syntactically legal but no warranty on semantics
- The erroneous symbol is attached to the appropriate person
- This associates an illegal action to the person via bad syntax substrings

Error Detection & Recovery example

- Following table shows three types of error detection and recovery for ‘aabc’:

(A)

Grammar		
S	$\rightarrow$	AB
A	$\rightarrow$	aa
	$\rightarrow$	aaA
B	$\rightarrow$	bc
	$\rightarrow$	bcB

(B)

Earley Chart		
predicted		
$0 : {}_{D}$	$\rightarrow$	$.S$
$0 : {}_{D}S$	$\rightarrow$	$.AB$
$0 : \_0A$	$\rightarrow$	$.aa$
$0 : {}_{D}A$	$\rightarrow$	$.aaA$
scanned "a"		
$1 : {}_{D}A$	$\rightarrow$	$a.a$
$1 : {}_{D}A$	$\rightarrow$	$a.aA$
none completed		
predicted		
$1 : {}_{1}A$	$\rightarrow$	$a.a$
$1 : {}_{1}A$	$\rightarrow$	$a.aA$

(C)

Insertion	Substitution	Deletion
scanned "b"	*scanned "b"*	*scanned "b"*
failed, expecting "a"	failed, expecting "a"	failed, expecting "a"
ignore "b"	*scanned "a"	*scanned "a"
predicted	$2 : {}_{1}A \rightarrow aa.$	$2 : {}_{1}A \rightarrow aa.$
$1 : {}_{1}A \rightarrow a.a$	$2 : {}_{1}A \rightarrow aa.A$	$2 : {}_{1}A \rightarrow aa.A$
$1 : {}_{1}A \rightarrow a.aA$	*completed*	*completed*
scanned "c"	$2 : {}_{1}A \rightarrow aa.$	$2 : {}_{1}A \rightarrow aa.$
failed, expecting "a"	$2 : {}_{D}S \rightarrow A.B$	$2 : {}_{D}S \rightarrow A.B$
TERMINATED	*predict*	*predict*
	$2 : {}_{2}A \rightarrow .aa$	$2 : {}_{2}A \rightarrow .aa$
	$2 : {}_{2}A \rightarrow .aaA$	$2 : {}_{2}A \rightarrow .aaA$
	$2 : {}_{2}B \rightarrow .bc$	$2 : {}_{2}B \rightarrow .bc$
	$2 : {}_{2}B \rightarrow .bcB$	$2 : {}_{2}B \rightarrow .bcB$
	scanned "c"	*scanned "b" (retry)
	failed, expecting "b"	$3 : {}_{2}B \rightarrow b.c$
	TERMINATED	$3 : {}_{2}B \rightarrow b.cB$
		none completed
		predicted
		$3 : {}_{3}B \rightarrow b.c$
		$3 : {}_{3}B \rightarrow b.cB$
		scanned "c"
		$3 : {}_{2}B \rightarrow b.c$
		$3 : {}_{2}B \rightarrow b.cB$
		$3 : {}_{2}B \rightarrow b.cB$

Error Detection & Recovery example (contd.)

- Error detection and recovery occurs by:
  - Maintaining every recovery path for multiple tracks
  - Tolerating only two consecutive failures
- Optimizations can be applied to the above method:
  - More consecutive failures can be tolerated by applying penalty to $P_D(a)$

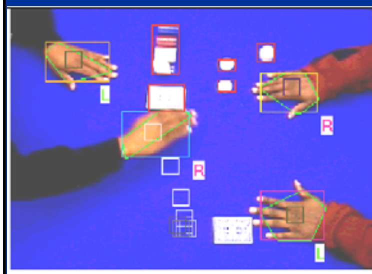
$$\hat{P}_D(a) = e^{\frac{-n}{\rho}} \widetilde{P}_D(a)$$

where, n=number of consecutive failures, ρ is empirically derived constant

  - Pruning recovered paths with low probability
  - Hybrid error scenarios by taking all three scenarios into account at each bad input

Experimental Results

- Experiment I: Event Detection Accuracy
  - Overall detection rate of events is 96.2% with 0.4% insertion, 0.1% substitution, 3.4% deletion errors
  - Player cheating can be detected by looking at 'c' and 'g' events



Vision system tracking the game.

S	Domain-Specific Events	Detect Rate	Error Rate (%)		
			Ins	*Sub*	*Del*
a	dlr removed house card	100.0	0.0	0.0	0.0
b	dlr removed plyr card	100.0	0.0	0.0	0.0
c	plyr removed house card†	100.0	0.0	0.0	0.0
d	plyr removed plyr card	100.0	0.0	0.0	0.0
e	dlr add card to house	94.6	0.0	0.0	5.4
f	dlr dealt card to plyr	92.2	0.0	0.0	7.8
g	plyr add card to house†	100.0	0.0	0.0	0.0
h	plyr add card to plyr	89.3	3.6	0.0	7.1
i	dlr removed chip	93.7	0.0	0.0	6.3
j	plyr removed chip	96.9	0.0	0.0	3.1
k	dlr pays plyr chip	96.9	0.0	0.0	3.1
l	plyr bet chip	90.5	1.1	1.1	7.4

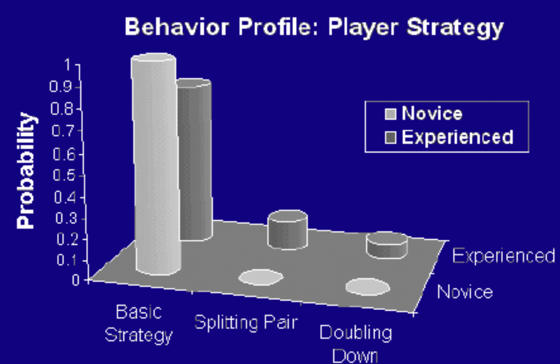
Experimental Results (contd.)

- Experiment II: Error Detection & Recovery
  - 100% accuracy with no insertion, substitution and deletion errors
  - Error recovery disabled:
    - Corpus A had 42.9% entire parses with 70.1% events detected
    - Error rates were insertion 5.8%, substitution 14.5% and deletion 9.6%
  - Error recovery enabled:
    - Corpus A had 85.7% entire parses with 93.8% events detected
    - Error rates reduced by insertion 70.5%, substitution 87.3% and deletion 71.9%

S	Detect %		Ins Err		Sub Err		Del Err	
	on	*off*	*on*	*off*	*on*	*off*	*on*	*off*
a	98.8	92.5	0.0	0.0	0.0	0.0	1.2	7.5
b	97.8	90.8	0.0	0.0	0.0	0.0	2.2	9.2
c	100.0	80.0	0.0	0.0	0.0	20.0	0.0	0.0
d	100.0	91.7	0.0	0.0	0.0	0.0	0.0	8.3
e	94.0	74.9	1.2	5.0	1.2	7.5	3.6	12.5
f	95.6	70.3	0.0	2.3	0.0	9.2	4.4	18.3
g	100.0	50.0	0.0	0.0	0.0	50.0	0.0	0.0
h	80.0	41.7	4.0	8.3	8.0	33.3	8.0	16.7
i	92.9	88.9	0.0	0.0	0.0	0.0	7.1	11.1
j	96.5	92.7	0.0	0.0	0.0	0.0	3.5	7.3
k	79.0	12.5	10.5	36.5	10.5	43.8	0.0	7.3
l	90.6	55.8	4.7	17.2	2.4	9.8	2.4	17.2

Experimental Results (contd.)

- Experiment III: High-level Behavior Assessment
  - 4 categories of behavior with 10 trials per individual to assess behavior



Behavior	Detection Accuracy
Low-risk	92%
High-risk	76%
Novice	100%
Expert	90%

Contributions

- Introduced SCFG for multitasked activities in separable groups
- Extended and improved error detection and recovery method in SCFG from Ivanov and Bobick's method (2000 PAMI)
- High-level behavior assessment from event sequence and frequency