Advanced Machine Learning & Perception

Instructor: Tony Jebara
Graphical (Structured) Models

• From Structured Prediction to Graphical Models
• Inference
• From Logic Networks to Bayesian Networks
• A Review of Graphical Models
• Junction Tree Algorithm
• MAP Estimation (ArgMax Junction Tree Algorithm)
• Loopy Propagation
Structured Prediction

• The key of structured prediction is fast computation of:

\[
\arg\max_{y \in Y} \ w^T \phi(x, y)
\]

• Usually, the space \( Y \) is too huge to enumerate
• But, if it has independencies, we can quickly find the max
• This is equivalent to finding the max of a graphical model

\[
p(y) = \frac{1}{Z} \exp \left( w^T \phi(x, y) \right)
\]

• The argmax of \( p(y) \) is the same as the argmax of above
• If \( y \) splits into many conditionally independent terms
  \( \rightarrow \) finding the max (Decoding) may be efficient
• Graphical models have three canonical problems to solve:
  1) Marginal inference, 2) Decoding and 3) Learning
Structured Prediction & HMMs

• Recall Hidden Markov Model (now y is observed, q hidden):

  \[ q_0 \xrightarrow{\pi} q_1 \xrightarrow{\alpha} q_2 \xrightarrow{\alpha} q_3 \xrightarrow{\alpha} q_4 \]

  space of q’s is \( O(M^T) \)

• Here, space of q’s is huge just like in structure prediction

• Would like to do 3 basic things with graphical models:
  1) Evaluate: given \( y_1, \ldots, y_T \) compute likelihood \( p(y_1, \ldots, y_T) \)
  2) Decode: given \( y_1, \ldots, y_T \) compute best \( q_1, \ldots, q_T \) or \( p(q_t) \)
  3) Learn: given \( y_1, \ldots, y_T \) learn parameters \( \theta \)

• Typically, HMMs use Baum-Welch, \( \alpha-\beta \) or Viterbi algorithm

• More general graphical models use Junction Tree Algorithm

• The JTA is a way of performing efficient inference
Inference

• Inference: goal is to predict some variables given others
  x1: flu
  x2: fever
  x3: sinus infection
  x4: temperature
  x5: sinus swelling
  x6: headache

  Patient claims headache and high temperature.
  Does he have a flu?

Given findings variables $X_f$ and unknown variables $X_u$
predict queried variables $X_q$

• Classical approach: truth tables (slow) or logic networks

• Modern approach: probability tables (slow) or Bayesian networks (fast belief propagation, junction tree algorithm)
Logic Nets to Bayesian Nets

1980’s expert systems & logic networks became popular

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>x1 v x2</th>
<th>x1 ^ x2</th>
<th>x1 -&gt; x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Problem: inconsistency, 2 paths can give different answers

Problem: rules are hard, instead use soft probability tables

\[ x_3 = x_1 \land x_2 \]

\[
p(x_3 | x_1, x_2)\]

These directed graphs are called Bayesian Networks
Directed Graphical Models

• Factorize a large (how big?) probability over several vars

\[ p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid p_{a_i}) = \prod_{i=1}^{n} p(x_i \mid \pi_i) \]

• Interpretation
  1: flu
  2: fever
  3: sinus infection
  4: temperature
  5: sinus swelling
  6: headache

\[ p(x_1, \ldots, x_6) = p(x_1) p(x_2 \mid x_1) p(x_3 \mid x_1) p(x_4 \mid x_2) p(x_5 \mid x_3) p(x_6 \mid x_2, x_5) \]

2^6 2^1 2^2 2^2 2^2 2^2 2^3
Undirected Graphical Models

• Probability for undirected is defined via Potential Functions which are more flexible than conditionals or marginals

\[ p(X) = p(x_1, \ldots, x_M) = \frac{1}{Z} \prod_C \psi(X_C) \]

\[ Z = \sum_X \prod_C \psi(X_C) \]

• Just a factorization of \( p(X) \), \( Z \) just normalizes the pdf
• Potential functions are positive functions of (not mutually exclusive) sub-groups of variables
• Potential functions are over complete sub-graphs or cliques \( C \) in the graph, clique is a set of fully-interconnected nodes
• Use maximal cliques, absorb cliques contained in larger \( \psi \)

\[
p(X) = \frac{1}{Z} \psi(x_1, x_2) \psi(x_2, x_3) \psi(x_3, x_4, x_5) \psi(x_4, x_5, x_6)
\]
Moralization

- Converts directed graph into undirected graph
- By moralization, marrying the parents:
  1) Connect nodes that have common children
  2) Drop the arrow heads to get undirected

\[ p(x_1) p(x_2 \mid x_1) p(x_3 \mid x_1) p(x_4 \mid x_2) p(x_5 \mid x_3) p(x_6 \mid x_2, x_5) \]
\[ \rightarrow \frac{1}{Z} \psi(x_1, x_2) \psi(x_1, x_3) \psi(x_2, x_4) \psi(x_3, x_5) \psi(x_2, x_5, x_6) \]

- Note: moralization resolves coupling due to marginalizing
- Moral graph is more general (loses some independencies)
Junction Trees

• Given moral graph want to build Junction Tree:
  each node is a clique \( (\psi) \) of variables in moral graph
  edges connect cliques of the potential functions
  unique path between nodes & root node (tree)
  between connected clique nodes, have separators \( (\phi) \)
  separator nodes contain intersection of variables

\[
p(X) = \frac{1}{Z} \psi(A, B, D) \psi(B, C, D) \psi(C, D, E)
\]
Triangulation

• Problem: imagine the following undirected graph

• Not a Tree!
• To ensure Junction Tree is a tree (no loops, etc.) before forming it must first **Triangulate** moral graph before finding the cliques...

• Triangulating gives more general graph (like moralization)
• Adds links to get rid of cycles or loops
• Triangulation: Connect nodes in moral graph such that no cycle of 4 or more nodes remains in the graph
Triangulation

- **Triangulation**: Connect nodes in moral graph such that no chordless cycles (no cycle of 4+ nodes remains)

  - So, *add links*, but many possible choices...
  - HINT: keep largest clique size small (for efficient JTA)
  - Chordless: no edges between successor nodes in cycle
  - Sub-optimal triangulations of moral graph are Polynomial
  - Triangulation that minimizes largest clique size is NP
  - But, OK to use a suboptimal triangulation (slower JTA...)
Triangulation

• **Triangulation:** Connect nodes in moral graph such that no chordless cycles (no cycle of 4+ nodes remains)

  - 1-cycle OK
  - 2-cycle OK
  - 3-cycle OK
  - 3-cycle OK
  - 3-cycle OK

  - **So, add links,** but many possible choices...
  - **HINT:** keep largest clique size small (for efficient JTA)
  - **Chordless:** no edges between successor nodes in cycle
  - **Sub-optimal triangulations of moral graph are Polynomial**
  - **Triangulation that minimizes largest clique size is NP**
  - **But, OK to use a suboptimal triangulation (slower JTA...)**
Running Intersection Property

- Junction Tree must satisfy **Running Intersection Property**
- RIP: On unique path connecting clique $V$ to clique $W$, all other cliques share nodes in $V \cap W$
Running Intersection Property

• Junction Tree must satisfy **Running Intersection Property**
• RIP: On unique path connecting clique \( V \) to clique \( W \), all other cliques share nodes in \( V \cap W \)

**HINT:** Junction Tree has largest total separator cardinality

\[
|\Phi| = |\phi(B, C)| + |\phi(C, D)| = 2 + 2
\]

\[
|\Phi| = |\phi(C, D)| + |\phi(D)| = 2 + 1
\]
Forming the Junction Tree

• Now need to connect the cliques into a Junction Tree
• But, must ensure Running Intersection Property
• Theorem: a valid (RIP) Junction Tree connection is one that maximizes the cardinality of the separators

\[
JT^* = \max_{\text{TREE STRUCTURES}} |\Phi| \\
= \max_{\text{TREE STRUCTURES}} \sum_s |\Phi(X_s)|
\]

• Use Kruskal’s algorithm:
  1) Init Tree with all cliques unconnected (no edges)
  2) Compute size of separators between all pairs
  3) Connect the two cliques with the biggest separator cardinality which doesn’t create a loop in current Tree (maintains Tree structure)
  4) Stop when all nodes are connected, else goto 3
Kruskal Example

- Start with unconnected cliques (after triangulation)

<table>
<thead>
<tr>
<th></th>
<th>ACD</th>
<th>BDE</th>
<th>CDF</th>
<th>DEH</th>
<th>DFGH</th>
<th>FGHI</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACD</td>
<td>-</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BDE</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CDF</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DEH</td>
<td>-</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFGH</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FGHI</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Junction Tree Probabilities

• We now have a valid Junction Tree!
• What does that mean?
• Recall probability for undirected graphs:
  \[ p(X) = p(x_1, \ldots, x_M) = \frac{1}{Z} \prod_C \psi(X_C) \]
• Can write junction tree as potentials of its cliques:
  \[ p(X) = \frac{1}{Z} \prod_C \tilde{\psi}(X_C) \]
• Alternatively: clique potentials over separator potentials:
  \[ p(X) = \frac{1}{Z} \frac{\prod_C \psi(X_C)}{\prod_S \phi(X_S)} \]
• This doesn’t change/do anything! Just less compact...
• Like *de-absorbing* smaller cliques from maximal cliques:
  \[ \tilde{\psi}(A, B, D) = \frac{\psi(A, B, D)}{\phi(B, D)} \]
  ...gives back original formula if \( \phi(B, D) \cong 1 \)
Junction Tree Algorithm

- Send message from each clique to its separators of what it thinks the submarginal on the separator is.
- Normalize each clique by incoming message from its separators so it agrees with them.

\[
V = \{A, B\} \quad S = \{B\} \quad W = \{B, C\}
\]

If agree:
\[
\sum_{V \setminus S} \psi_V = \phi_s = p(S) = \phi_s = \sum_{W \setminus S} \psi_W
\]

Else:

Send message from V to W...
\[
\phi_s^* = \sum_{V \setminus S} \psi_V
\]
\[
\psi_w^* = \frac{\phi_s^*}{\phi_s} \psi_w
\]
\[
\psi_v^* = \psi_v
\]

Send message from W to V...
\[
\phi_s^{**} = \sum_{W \setminus S} \psi_w^{**}
\]
\[
\psi_v^{**} = \frac{\phi_s^{**}}{\phi_s^{*}} \psi_v^{*}
\]
\[
\psi_w^{**} = \psi_w^{*}
\]

Now they agree...Done!
\[
\sum_{V \setminus S} \psi_V^{**} = \sum_{V \setminus S} \frac{\phi_s^{**}}{\phi_s^*} \psi_V^{*}
\]
\[
= \frac{\phi_s}{\phi_s^*} \sum_{V \setminus S} \psi_V^{*}
\]
\[
= \phi_s = \sum_{W \setminus S} \psi_W^{**}
\]
Junction Tree Algorithm

- When “Done”, all clique potentials are marginals and all separator potentials are submarginals!
- Note that \( p(X) \) is unchanged by message passing step:

\[
\phi^*_S = \sum_{V \setminus S} \psi_V
\]
\[
\psi^*_W = \frac{\phi^*_S}{\phi_S} \psi_W
\]
\[
\psi^*_V = \psi_V
\]

\[
p(X) = \frac{1}{Z} \frac{\psi_V^* \psi_W^*}{\phi_S^*} = \frac{1}{Z} \frac{\psi_V \phi^*_S \psi_W}{\phi_S^*} = \frac{1}{Z} \frac{\psi_V \psi_W}{\phi_S}
\]

- Example: if potentials are poorly initialized... get corrected!

\[
\psi_{AB} = p(B \mid A)p(A)
= p(A, B)
\]
\[
\psi_{BC} = p(C \mid B)
\]
\[
\phi^*_B = \sum_A \psi_{AB} = \sum_A p(A, B) = p(B)
\]
\[
\psi^*_{BC} = \frac{\phi^*_S}{\phi_S} \psi_{BC} = \frac{p(B)}{1} p(C \mid B) = p(B, C)
\]
Junction Tree Algorithm

• Example: if evidence is observed... i.e. random var \( A := 1 \)

Initialize as before, cliques get underlying conditionals...

\[
\psi_{AB} = p(A, B) \quad \psi_{BC} = p(C \mid B) \quad \phi_B = 1
\]

Update with slice...

\[
\phi_B^* = \sum_A \psi_{AB} \delta(A = 1) = \sum_A p(A, B) \delta(A = 1) = p(A = 1, B)
\]

\[
\psi_{BC}^* = \frac{\phi_S}{\phi_S^*} \psi_{BC} = \frac{p(A = 1, B)}{1} p(C \mid B) = p(A = 1, B, C)
\]

\[
\psi_{AB}^* = \psi_{AB} = p(A = 1, B)
\]

To get conditionals...

\[
p(B, C \mid A = 1) = \frac{\psi_{BC}^*}{\sum_{B, C} \psi_{BC}^*}
\]

• Problem: if send message to neighbor & he changes, we must re-update! Could keep looping for a long time.
JTA: Collect & Distribute

- Trees: recursive, no need to reiterate messages mindlessly!
- Send a message only after hearing from all neighbors...

initialize(DAG){
    Pick root
    Set all variables as:
    \[
    \psi_C = p(x_i | \pi_i) \quad \forall i
    \]
    \[
    \phi^*_S = 1 \quad \forall S
    \]
    \[
    Z^S = 1
    \]
}

collectEvidence(node) {
    for each child of node {
        update(node,collectEvidence(child));
    }
    return(node);
}

distributeEvidence(node) {
    for each child of node {
        update(child,node);
        distributeEvidence(child);
    }
}

update(node,evidence) {
    \[
    \psi_C^* = \frac{\phi^*_S}{\sum_{C \setminus S} \psi_C} \psi_C
    \]
}
Junction Tree Algorithm

- JTA: 1) Initialize 2) Collect 3) Distribute
ArgMax Junction Tree Algorithm

• We can also use JTA for finding the max not the sum over the joint to get argmax of marginals & conditionals
• Say have some evidence:
  $$p(X_F, \bar{X}_E) = p(x_1, \ldots, x_n, \bar{x}_{n+1}, \ldots, \bar{x}_N)$$
• Most likely (highest p) $X_F$?
  $$X^*_F = \arg\max_{X_F} p(X_F, \bar{X}_E)$$
• What is most likely state of patient with fever & headache?
  $$p_F^* = \max_{x_2,x_3,x_4,x_5} p(x_1 = 1, x_2, x_3, x_4, x_5, x_6 = 1)$$
  $$= \max_{x_2} p(x_2 \mid x_1 = 1) p(x_1 = 1) \max_{x_3} p(x_3 \mid x_1 = 1)$$
  $$\max_{x_4} p(x_4 \mid x_2) \max_{x_5} p(x_5 \mid x_3) p(x_6 = 1 \mid x_2, x_5)$$
• Solution: update in JTA uses max instead of sum:
  $$\phi_S^* = \max_{V \setminus S} \psi_V \quad \psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W \quad \psi_V^* = \psi_V$$
• Final potentials aren’t marginals:
  $$\psi(X_C) = \max_{U \setminus C} p(X)$$
• Highest value in potential is most likely:
  $$X_C^* = \arg\max_C \psi(X_C)$$
Loopy Belief Propagation

- We could run junction tree algorithm on non-trees... but...
  - a) no guaranteed convergence
  - b) might get inexact marginals
  - c) might iterate indefinitely (not polynomial time)
- Called Loopy Propagation since messages loop indefinitely
- Example: Markov random field for images...

Just find cliques
Don't triangulate
Keep iterating JTA...
Sometimes Guaranteed!