CPSC 540: Machine Learning
Hidden Markov Models

Mark Schmidt
University of British Columbia
Winter 2019
Decoding in density models: finding $x$ with highest joint probability:

$$\arg\max_{x_1, x_2, \ldots, x_d} p(x_1, x_2, \ldots, x_d).$$

For Markov chains, we find decoding by writing maximization as

$$\max_{x_1, x_2, x_3, x_4} p(x_1, x_2, x_3, x_4) = \max_{x_4} \max_{x_3} p(x_4 \mid x_3) \max_{x_2} p(x_3 \mid x_2) \max_{x_1} p(x_2 \mid x_1) p(x_1),$$

The $M_j(x_j)$ functions are called messages (summarize everything about past).

Viterbi decoding computes $M_1(x_1)$ for all $x_1$, $M_2(x_2)$ for all $x_2$, and so on.
Chapman-Kolmogorov Equations as Message Passing

- We can also view Chapman Kolmogorov equations as message passing:

\[
\sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} p(x_1, x_2, x_3, x_4) = \sum_{x_4} \sum_{x_3} \sum_{x_2} \sum_{x_1} p(x_4 \mid x_3) p(x_3 \mid x_2) p(x_2 \mid x_1) p(x_1)
\]

\[
= \sum_{x_4} \sum_{x_3} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1) M_1(x_1)
\]

\[
= \sum_{x_4} \sum_{x_3} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) M_2(x_2)
\]

\[
= \sum_{x_4} \sum_{x_3} p(x_4 \mid x_3) M_3(x_3)
\]

\[
= \sum_{x_4} M_4(x_4),
\]

- Messages \(M_j(x_j)\) are the marginals of the Markov chain.
  - So we can view CK equations as Viterbi decoding with “max” replace by “sum”.
  - This two methods are also known as “max-product” and “sum-product” algorithms.
Message-Passing Algorithms

We’ve discussed several algorithms with similar structure:
- Viterbi decoding algorithm for decoding in discrete Markov chains.
- CK equations for marginals in discrete Markov chains.
- Gaussian updates for marginals in Gaussian Markov chains.

These are all special cases of message-passing algorithms:
1. Define $M_j$ summarizing all relevant information about the past at time $j$.
2. Use Markov property to write $M_j$ recursively in terms of $M_{j-1}$.
3. Solve task by computing $M_1, M_2, \ldots, M_d$.

“Generalized distributive law” is a framework for describing when/why this works:
- https://authors.library.caltech.edu/1541/1/AJIieeetit00.pdf

In some cases we’ll also need backwards message $V_j$ (“cost to go”):
- $V_j$ summarizes all relevant information about the future at time $j$. 
Backwards “Cost to Go” Messages

- Using backwards messages $V_j(x_j)$:

$$
\sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} p(x_1, x_2, x_3, x_4) = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)
$$

$$
= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1) \sum_{x_3} p(x_3 \mid x_2) \sum_{x_4} p(x_4 \mid x_3)
$$

$$
= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1) \sum_{x_3} p(x_3 \mid x_2) \sum_{x_4} p(x_4 \mid x_3) V_4(x_4)
$$

$$
= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1) \sum_{x_3} p(x_3 \mid x_2) V_3(x_3)
$$

$$
= \sum_{x_1=c} p(x_1) \sum_{x_2} p(x_2 \mid x_1) V_2(x_2)
$$

$$
= \sum_{x_1} p(x_1) V_1(x_1).
$$

- Observe that backwards messages $V_j(x_j)$ are not probabilities as in CK equations.
  - But they summarize everything you need to know about the future.
Computing all $M_j(x_j)$ and $V_j(x_j)$ is called the forward-backward algorithm. Not interesting for Markov chains since $V_j(x_j) = 1$ for all $j$ and $x_j$.

Why do we need both types of messages?
- Can efficiently compute all conditionals $p(x_j = s \mid x_{10} = 3)$ (for all $j$ and $s$).
  - Messages are modified when you condition (see bonus slides).
  - The modified $V_j(x_j)$ will reflect “what you need to know about the future events”.

- Can be used to compute probabilities in generalizations of Markov chains (next).
  - In this setting the forward messages may not be probabilities either.

- In reinforcement learning, estimating the “cost to go” (“value”) function is the goal.
  - We aren’t covering RL, but understanding Markov chains will help you understand RL.
Back to the Rain Data

- We previously considered the “Vancouver Rain” data:

- We said that a **homogeneous Markov chain** is a good model:
  - Captures direct dependency between adjacent days.
Back to the Rain Data

- But doesn't it rain less in the summer?

- There are hidden clusters in the data not captured by the Markov chain.  
  - But mixture of independent models are inefficient at representing direct dependency.

- Mixture of Markov chains could capture direct dependence and clusters,

\[
p(x_1, x_2, \ldots, x_d) = \sum_{c=1}^{k} p(z = c) p(x_1 | z = c) p(x_2 | x_1, z = c) \cdots p(x_d | x_{d-1}, z = c). \]

- Cluster \( z \) chooses which homogeneous Markov chain parameters to use.
  - We could learn that we're more likely to have rain in winter.
  - Can modify CK equations to take into account \( z \), and then apply EM.
Back to the Rain Data

- The rain data is artificially divided into months.

- We previously discussed viewing rain data as one very long sequence ($n = 1$).

- We could apply homogeneous Markov chains due to parameter tying.

- But a mixture doesn’t make sense when $n = 1$.

- What we want: different “parts” of the sequence come from different clusters.
  - We transition from “summer” cluster to “fall” cluster at some time $j$.

- One way to address this (“hidden” Markov model):
  - Let each day have its own cluster.
  - Have a Markov dependency between cluster values of adjacent days.
Hidden Markov Models

- **Hidden Markov models** have each $x_j$ depend on hidden Markov chain.

  $p(x_1, x_2, \ldots, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j | z_{j-1}) \prod_{j=1}^{d} p(x_j | z_j)$.

- We’re going to learn clusters $z_j$ and the hidden dynamics.
  - Hidden cluster $z_j$ could be “summer” or “winter” (we’re learning the clusters).
  - Transition probability $p(z_j | z_{j-1})$ is probability of staying in “summer”.
  - Emission probability $p(x_j | z_j)$ is probability of “rain” during “summer”.

![Directed Acyclic Graphical Models](image-url)
Hidden Markov Models

- Hidden Markov models have each $x_j$ depend on hidden Markov chain.

\[
p(x_1, x_2, \ldots, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j).
\]

- You observe the $x_j$ values but do not see the $z_j$ values.
  - CK equations won’t work since $p(z_1 = s)$ depends on future $x_j$ values.

- But forward-backward algorithm can be used to compute probabilities.
  - And subsequently learn with EM.
Hidden Markov Models

- **Hidden Markov models** have each $x_j$ depend on hidden Markov chain.

  ![Diagram of hidden Markov models]

- Note that the $x_j$ can be continuous even with discrete clusters $z_j$.
  - You could have a “mixture of Gaussians” with cluster changing in time.

- If the $z_j$ are continuous it’s often called a state-space model.
  - If everything is Gaussian, it leads to Kalman filtering.
  - Keywords for non-Gaussian: unscented Kalman filter and particle filter.

- Variants of HMMs are probably the most-used time-series model...
Applications of HMMs and Kalman Filters

HMMs can be applied in many fields where the goal is to recover a data sequence that is not immediately observable (but other data that depend on the sequence are).

Applications include:
- Single Molecule Kinetic analysis
- Cryptanalysis
- Speech recognition
- Speech synthesis
- Part-of-speech tagging
- Document Separation in scanning solutions
- Machine translation
- Partial discharge
- Gene prediction
- Alignment of bio-sequences
- Time Series Analysis
- Activity recognition
- Protein folding
- Metamorphic Virus Detection
- DNA Motif Discovery

Applications
- Attitude and Heading Reference Systems
- Autopilot
- Battery state of charge (SoC) estimation
- Brain-computer interface
- Chaotic signals
- Tracking and Vertex Fitting of charged particles in Particle Detectors
- Tracking of objects in computer vision
- Dynamic positioning
- Economics, in particular macroeconomics, time series analysis, and econometrics
- Inertial guidance system
- Orbit Determination
- Power system state estimation
- Radar tracker
- Satellite navigation systems
- Seismology
- Sensorless control of AC motor variable-frequency drives
- Simultaneous localization and mapping
- Speech enhancement
- Visual odometry
- Weather forecasting
- Navigation system
- 3D modeling
- Structural health monitoring
- Human sensorimotor processing
Example: Modeling DNA Sequences

- A nice demo of independent vs. Markov vs. HMMs for DNA sequences:

- Independent model for elements of sequence:

```plaintext
p_A=0.2, p_C=0.3, p_G=0.3, p_T=0.2
```
Example: Modeling DNA Sequences

- **Markov model** for elements of sequence (dependence on previous symbol):

![Diagram](image)

- "AfterA" wheel
  - $p_A=0.2$, $p_C=0.3$, $p_G=0.3$, $p_T=0.2$

- "AfterC" wheel
  - $p_A=0.1$, $p_C=0.41$, $p_G=0.39$, $p_T=0.1$

- "AfterG" wheel
  - $p_A=0.25$, $p_C=0.25$, $p_G=0.25$, $p_T=0.25$

- "AfterT" wheel
  - $p_A=0.5$, $p_C=0.17$, $p_G=0.17$, $p_T=0.17$
Example: Modeling DNA Sequences

- **Hidden Markov model (HMM)** for elements of sequence:
  
  ![Diagram](image)
  
  - Can reflect that *probabilities are different in different regions*.
  - You probably get a better model by consider hidden Markov and visible Markov.
    - Would have “treewidth 2”, which we’ll show later means it’s tractable to use.
Who is Guarding Who?

- There is a lot of data on offense of NBA basketball players.
  - Every point and assist is recorded, more scoring gives more wins and $$\$$.  

- But how do we measure defense?
  - We need to know who each player is guarding.

- HMMs can be used to model who is guarding who over time.
  - [https://www.youtube.com/watch?v=JvNkZdZJBt4](https://www.youtube.com/watch?v=JvNkZdZJBt4)
Outline

1. Hidden Markov Models
2. Directed Acyclic Graphical Models
Higher-Order Markov Models

- **Markov models** use a density of the form

  \[
p(x) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3) \cdots p(x_d \mid x_{d-1}).
\]

- They support efficient computation but **Markov assumption is strong**.

- A more flexible model would be a **second-order Markov model**,  

  \[
p(x) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2, x_1)p(x_4 \mid x_3, x_2) \cdots p(x_d \mid x_{d-1}, x_{d-2}),
\]

  or even a higher-order models.

- General case is called **directed acyclic graphical (DAG) models**:  
  - They allow dependence on any subset of previous features.
DAG Models

- As in Markov chains, DAG models use the chain rule to write

  \[ p(x_1, x_2, \ldots, x_d) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_d \mid x_1, x_2, \ldots, x_{d-1}). \]

- We can alternately write this as:

  \[ p(x_1, x_2, \ldots, x_d) = \prod_{j=1}^{d} p(x_j \mid x_{1:j-1}). \]

- In Markov chains, we assumed \( x_j \) only depends on previous \( x_{j-1} \) given past.

- In DAGs, \( x_j \) can depend on any subset of the past \( x_1, x_2, \ldots, x_{j-1} \).
We often write joint probability in DAG models as

\[ p(x_1, x_2, \ldots, x_d) = \prod_{j=1}^{d} p(x_j \mid x_{\text{pa}(j)}) , \]

where \( \text{pa}(j) \) are the “parents” of node \( j \).

- For Markov chains the only “parent” of \( j \) is \( (j - 1) \).
- If we have \( k \) parents we only need \( 2^{k+1} \) parameters.

This corresponds to a set of conditional independence assumptions,

\[ p(x_j \mid x_{1:j-1}) = p(x_j \mid x_{\text{pa}(j)}) , \]

that we’re independent of previous non-parents given the parents.
Recall trying to model digits using an inhomogeneous Markov chain:

Only models dependence on pixel above, not on 2 pixels above nor across columns.
MNIST Digits with DAG Model (Sparse Parents)

- Samples from a DAG model with 8 parents per feature:

Parents of \((i, j)\) are 8 other pixels in the neighbourhood (“up by 2, left by 2”):
\[
\{(i-2, j-2), (i-1, j-2), (i, j-2), (i-2, j-1), (i-1, j-1), (i, j-1), (i-2, j), (i-1, j)\}.
\]
From Probability Factorizations to Graphs

- DAG models are also known as "Bayesian networks" and "belief networks".

- "Graphical" name comes from visualizing parents/features as a graph:
  - We have a node for each feature $j$.
  - We place an edge into $j$ from each of its parents.

- The DAG representation for a Markov chains is:

```
X_1 -> X_2 -> X_3 -> X_4 -> X_5
```

- Different than "state transition diagrams": edges are between variables (not states).

- This graph is not just a visualization tool:
  - Can be used to test arbitrary conditional independences ("d-separation").
  - Graph structure tells us whether message passing is efficient ("treewidth").
Graph Structure Examples

With product of independent we have

\[ p(x) = \prod_{j=1}^{d} p(x_j), \]

so \( \text{pa}(j) = \emptyset \) and the graph is:
With Markov chain we have

\[ p(x) = p(x_1) \prod_{j=2}^{d} p(x_j \mid x_{j-1}), \]

so \( \text{pa}(j) = \{j - 1\} \) and the graph is:

\[ X_1 \xrightarrow{} X_2 \xrightarrow{} X_3 \xrightarrow{} X_4 \xrightarrow{} X_5 \]
Graph Structure Examples

With second-order Markov chain we have

\[
p(x) = p(x_1)p(x_2 | x_1) \prod_{j=3}^{d} p(x_j | x_{j-1}, x_{j-2}),
\]

so \( \text{pa}(j) = \{j - 2, j - 1\} \) and the graph is:
With **general distribution** we have

\[ p(x) = \prod_{j=1}^{d} p(x_j \mid x_{1:j-1}). \]

so \( \text{pa}(j) = \{1, 2, \ldots, j - 1\} \) and the graph is:
Graph Structure Examples

In naive Bayes (or GDA with diagonal $\Sigma$) we add an extra variable $y$ and use

$$p(y, x) = p(y) \prod_{j=1}^{d} p(x_j \mid y),$$

which has $\text{pa}(y) = \emptyset$ and $\text{pa}(x_j) = y$ giving

![Graph Diagram]
Graph Structure Examples

With mixture of independent models we have

\[
p(z, x) = p(z) \prod_{j=1}^{d} p(x_j | z).
\]

which has pa\((z) = \emptyset\) and pa\((x_j) = z\) giving same structure as naive Bayes:

Since structure is the same, many computations will be similar.
Graph Structure Examples

With mixture of Markov chains models we have

\[ p(x_1, x_2, \ldots, x_d, z) = p(z)p(x_1 \mid z) \prod_{j=2}^{d} p(x_j \mid x_{j-1}, z). \]

which has \( \text{pa}(z) = \emptyset \) and \( \text{pa}(x_j) = \{x_{j-1}, z\} \):
Graph Structure Examples

Sometimes it’s easier to present a model using the graph.

In hidden Markov models we have this structure:

The graph and variable names already give you an idea of what this model does:
- We have hidden variables $z_j$ that follow a Markov chain.
- Each feature $x_j$ depends on corresponding hidden variable $z_j$. 
Graph Structure Examples

- Instead of factorizing by variables $j$, could factor into blocks $b$:

$$p(x) = \prod_b p(x_b \mid x_{pa(b)}),$$

and have the nodes be blocks (usually assuming **full connectivity within the block**).

- With **mixture of Gaussian** and full covariances we have

$$p(z, x) = p(z)p(x \mid z).$$

- The corresponding graph structure is:

- **Gaussian generative classifiers** (GDA) have the same structure.
With probabilistic PCA we have

\[ p(z, x) = p(x \mid z) \prod_{c=1}^{k} p(z_c). \]

The corresponding graph structure is:

The data \( x \) comes from a set of independent parents (latent factors).
Graph Structure Examples

We can consider less-structured examples,

The corresponding factorization is:

\[ p(S, V, R, W, G, D) = p(S)p(V)p(R | V)p(W | S, R)p(G | V)p(D | G). \]
Graph Structure Examples

We can consider **phylogeny** (family trees):
Summary

- **Message-passing** allow efficient calculations with Markov chains.

- **Hidden Markov models** model time-series with hidden per-time cluster.
  - Tons of applications, typically more realistic than Markov models.

- **DAG models** factorize joint distribution into product of conditionals.
  - Assume conditionals depend on small number of “parents”.
  - Joint distribution of models we’ve discussed can be written as DAG models.

- Next time: the IID assumption as a graphical model?
Computing Conditional Probabilities

- Previously: Monte Carlo for approximating conditional probabilities
- For Gaussian/discrete Markov chains, we can do better than rejection sampling.
  1. We can generate exact samples from conditional distribution (bonus slide).
     - Rejection sampling is not needed, relies on “backwards sampling” in time.
  2. We can find conditional decoding $\max_x | x_j = c \ p(x)$:
     - Run Viterbi decoding with $M_{j'}(c) = 1$ and $M_{j'}(c') = 0$ for $c \neq c'$.
  3. We can find univariate conditionals, $p(x_j | x_{j'})$.

- Example of computing $p(x_1 = c | x_3 = 1)$ in a length-4 discrete Markov chain:

  
  $$p(x_1 = c | x_3 = 1) \propto p(x_1 = c, x_3 = 1) = \sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4),$$

  where the normalizing constant is the marginal $p(x_3 = 1)$.
- This is a sum over $k^{d-2}$ possible assignments to other variables.
Distributing Sum across Product

Fortunately, the **Markov property** makes the sums simplify as before:

\[
\sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) = \sum_{x_4} \sum_{x_3=1} \sum_{x_2} \sum_{x_1=c} p(x_4 | x_3)p(x_3 | x_2)p(x_2 | x_1)p(x_1) \\
= \sum_{x_4} \sum_{x_3=1} \sum_{x_2} p(x_4 | x_3)p(x_3 | x_2) \sum_{x_1=c} p(x_2 | x_1)p(x_1) \\
= \sum_{x_4} \sum_{x_3=1} p(x_4 | x_3) \sum_{x_2} p(x_3 | x_2) \sum_{x_1=c} p(x_2 | x_1)M_1(x_1) \\
= \sum_{x_4} \sum_{x_3=1} p(x_4 | x_3) \sum_{x_2} p(x_3 | x_2)M_2(x_2) \\
= \sum_{x_4} \sum_{x_3=1} p(x_4 | x_3)M_3(x_3) \\
= \sum_{x_4} M_4(x_4),
\]

where \( M_j(x_j) \) now sums over paths ending in \( x_j \) instead of maximizing.

- And we set \( M_1(c') = 0 \) if \( c' \neq c \) and \( M_3(c') = 0 \) for \( c' \neq 1 \).
Performing our conditional calculation using backwards messages.

\[
\sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) = \sum_{x_1 = c} \sum_{x_2} \sum_{x_3 = 1} \sum_{x_4} p(x_4 | x_3) p(x_3 | x_2) p(x_2 | x_1) p(x_1)
\]

\[
= \sum_{x_1 = c} p(x_1) \sum_{x_2} p(x_2 | x_1) \sum_{x_3 = 1} p(x_3 | x_2) \sum_{x_4} p(x_4 | x_3)
\]

\[
= \sum_{x_1 = c} p(x_1) \sum_{x_2} p(x_2 | x_1) \sum_{x_3 = 1} p(x_3 | x_2) \sum_{x_4} p(x_4 | x_3) V_4(x_4)
\]

\[
= \sum_{x_1 = c} p(x_1) \sum_{x_2} p(x_2 | x_1) \sum_{x_3 = 1} p(x_3 | x_2) V_3(x_3)
\]

\[
= \sum_{x_1 = c} p(x_1) \sum_{x_2} p(x_2 | x_1) V_2(x_2)
\]

\[
= \sum_{x_1 = c} p(x_1) V_1(x_1).
\]
Generic forward and backward messages for discrete marginals have the form

\[ M_j(x_j) = \sum_{x_{j-1}} p(x_j \mid x_{j-1}) M_{j-1}(x_{j-1}), \quad V_j(x_j) = \sum_{x_{j+1}} p(x_{j+1} \mid x_j) V_{j+1}(x_{j+1}). \]

We can compute \( p(x_j = c \mid x_j' = c') \) using only forward messages:

- Set \( M_j(c) = 1 \) and \( M_{j'}(c') = 1 \).

Why we would need backward messages?
We can compute $p(x_j = c \mid x_{j'} = c')$ for all $j$ in $O(dk^2)$ with both messages.

First compute all message normally with $M_{j'}(c') = 1$ and $V_{j'}(c') = 1$. (Other $M_{j'}$ and $V_{j'}$ are set to 0)

We then have that

- $M_j(x_j)$ sums up all the paths that end in state $x_j$ (with $x_{j'} = c'$).
- $V_j(x_j)$ sums up all the paths that start in state $x_j$ (with $x_{j'} = c'$).
- We can combine these values to get
  
  $$p(x_j \mid x_{j'}) \propto M_j(x_j)V_j(x_j),$$

- Computing all $M_j$ and $V_j$ is called the **forward-backward algorithm**.
Conditional Samples from Gaussian/Discrete Markov Chain

Generating exact conditional samples from Gaussian/discrete Markov chains:

1. If we’re only conditioning on first $j$ states, $x_{1:j}$, just fix these values and start ancestral sampling from time $(j + 1)$.

2. If we have the marginals $p(x_j)$, we can get the “backwards” transition probabilities using Bayes rule,

$$p(x_j \mid x_{j+1}) = \frac{p(x_{j+1} \mid x_j)p(x_j)}{p(x_{j+1})},$$

which lets us run ancestral sampling in reverse: sample $x_d$ from $p(x_d)$, then $x_{d-1}$ from $p(x_{d-1} \mid x_d)$, and so on.

3. If we’re only conditioning on last $j$ states $x_{d-j:d}$, run CK equations to get marginals and then start ancestral sampling “backwards” starting from $(d - j - 1)$ to sample the earlier states.
If we’re conditioning on contiguous states in the middle, \( x_j : j' \), run ancestral sampling forward starting from position \((j' + 1)\) and backwards starting from position \((j - 1)\).

If you condition on non-contiguous positions \( j \) and \( j' \) with \( j < j' \), need to do (i) forward sampling starting from \((j' + 1)\), (ii) backward sampling starting from \((j - 1)\), and (iii) CK equations on the sequence \((j : j')\) to get marginals conditioned on value of \( j \) then backwards sampling back to \( j \) starting from \((j' - 1)\).

The above are all special cases of conditioning in an undirected graphical model (UGM), followed by applying the “forward-filter backward-sampling” algorithm on each of the resulting chain-structured UGMs.