CPSC 440/540: Advanced Machine Learning

Message Passing; MCMC

Danica Sutherland (building on materials from Mark Schmidt)

University of British Columbia

Winter 2023

Last Time: Markov Chains

- State space, initial probabilities, transition matrix
- Homogeneous or inhomogeneous
- MLE: just fit appropriate categorical distribution (by counting) for each part
- Inference: ancestral sampling, marginals with CK equations

Application: Voice Photoshop

bonus!

- Adobe VoCo uses decoding in a Markov chain as part of synthesizing voices:

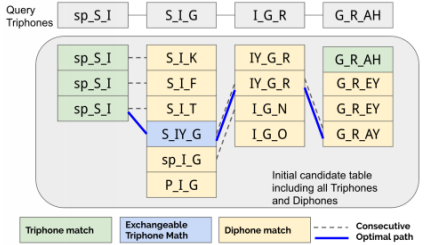


Fig. 7. Dynamic triphone preselection. For each query triphone (top) we find a candidate set of good potential matches (columns below). Good paths through this set minimize differences from the query, number and severity of breaks, and contextual mismatches between neighboring triphones.

http://gfx.cs.princeton.edu/pubs/Jin_2017_VTI/Jin2017-VoCo-paper.pdf

- https://www.youtube.com/watch?v=I3l4XLZ59iw

Decoding: Maximizing Joint Probability

- Decoding the mode in density models: finding x with highest joint probability:

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

- For CS grad student ($d = 60$) the mode is `industry` for all years.
  - The mode often doesn't look like a typical sample.
  - The mode can change if you increase d.

- Decoding is easy for independent models:
  - Here, $p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2)p(x_3)p(x_4)$.
  - You can optimize $p(x_1, x_2, x_3, x_4)$ by optimizing each $p(x_j)$ independently.

- Can we also maximize the marginals to decode a Markov chain?

Example of Decoding vs. Maximizing Marginals

- Consider the "plane of doom" 2-variable Markov chain:

$$X = \begin{bmatrix} \texttt{land} & \texttt{alive} \\ \texttt{land} & \texttt{alive} \\ \texttt{crash} & \texttt{dead} \\ \texttt{explode} & \texttt{dead} \\ \texttt{crash} & \texttt{dead} \\ \texttt{land} & \texttt{alive} \\ \vdots & \vdots \end{bmatrix}.$$

- 40% of the time the plane lands and you live.
- 30% of the time the plane crashes and you die.
- 30% of the time the explodes and you die.

Example of Decoding vs. Maximizing Marginals

- Initial probabilities are given by

$$\Pr(x_1 = \texttt{land}) = 0.4, \quad \Pr(x_1 = \texttt{crash}) = 0.3, \quad \Pr(x_1 = \texttt{explode}) = 0.3,$$

  and transition probabilites are:

$$\Pr(X_2 = \texttt{alive} \mid X_1 = \texttt{land}) = 1, \quad \Pr(X_2 = \texttt{alive} \mid X_1 = \texttt{crash}) = 0,$$
$$\Pr(X_2 = \texttt{alive} \mid X_1 = \texttt{explode}) = 0.$$

- From the CK equations, we know

$$\Pr(X_2 = \texttt{alive}) = 0.4, \quad \Pr(X_2 = \texttt{dead}) = 0.6$$

- Maximizing the marginals $p(x_j)$ independently gives (land, dead).
  - This has probability 0, since $\Pr(\texttt{dead} \mid \texttt{land}) = 0$.

- Decoding considers the joint assignment to x_1 and x_2 maximizing probability.
  - In this case it's (land, alive), which has probability 0.4.

Decoding with Dynamic Programming

- Note that decoding can't be done forward in time as in CK equations.
  - Even if $\Pr(x_1 = 1) = 0.99$, the most likely sequence could have $x_1 = 2$.
  - So we need to optimize over all k^d assignments to all variables.
- Fortunately, we can solve this problem using dynamic programming.
- Ingredients of dynamic programming:
  1. Optimal sub-structure.
     - We can divide the problem into sub-problems that can be solved individually.
  2. Overlapping sub-problems.
     - The same sub-problems are reused several times.

Decoding with Dynamic Programming

- For decoding in Markov chains, we'll use the following sub-problem:
  - Compute the highest probability sequence of length j ending in state c.
  - We'll use $M_j(c)$ as the probability of this sequence.

$$M_j(c) = \max_{x_1, x_2, \ldots, x_{j-1}} p(x_1, x_2, \ldots, x_{j-1}, c).$$

- Optimal sub-structure:
  - We can find the decoding by taking $\arg\max_{x_d} M_d(x_d)$, then backtracking.
  - Base case: $M_1(c) = \Pr(X_1 = c)$, which we're given.
  - We can compute other $M_j(s)$ recursively (derivation of this coming up),

$$M_j(s) = \max_{x_{j-1}} \underbrace{\Pr(x_j = c \mid X_{j-1} = x_{j-1})}_{\text{given}} \underbrace{M_{j-1}(x_{j-1})}_{\text{recurse}}.$$

- Overlapping sub-problems:
  - The same k values of $M_{j-1}(s)$ are used to compute the k values of $M_j(s)$.

Digression: Recursive Joint Maximization

- To derive the M_j formula, it will be helpful to re-write joint maximizations as

$$\max_{x_1,x_2} f(x_1,x_2) = \max_{x_1} g(x_1) \quad \text{where} \quad g(x_1) = \max_{x_2} f(x_1,x_2).$$

  - This f_1 "maximizes out" x_2, similar to marginalization rule in probability.

- Can also write this as

$$\max_{x_1,x_2} f(x_1,x_2) = \max_{x_1} \underbrace{\max_{x_2} f(x_1,x_2)}_{g(x_1)}.$$

- You can do this trick repeatedly and/or with any number of variables.

Decoding with Dynamic Programming

- Derivation of recursive calculation for $M_j(x_j)$ for decoding Markov chains:

$$
\begin{aligned}
M_j(x_j) &= \max_{x_1,x_2,\ldots,x_{j-1}} p(x_1, x_2, \ldots, x_j) && \text{(definition of } M_j(x_j)\text{)} \\
&= \max_{x_1,x_2,\ldots x_{j-1}} p(x_j \mid x_1, x_2, \ldots x_{j-1}) p(x_1, x_2, \ldots, x_{j-1}) && \text{(product rule)} \\
&= \max_{x_1,x_2,\ldots x_{j-1}} p(x_j \mid x_{j-1}) p(x_1, x_2, \ldots, x_{j-1}) && \text{(Markov property)} \\
&= \max_{x_{j-1}} \left\{ \max_{x_1,x_2,\ldots x_{j-2}} p(x_j \mid x_{j-1}) p(x_1, x_2, x_{j-1}) \right\} && (\max_{a,b} f(a,b) = \max_a \{\max_b f(a,b)\}) \\
&= \max_{x_{j-1}} \left\{ p(x_j \mid x_{j-1}) \max_{x_1,x_2,\ldots x_{j-2}} p(x_1, x_2, x_{j-1}) \right\} && (\max_i \alpha a_i = \alpha \max_i a_i \text{ for } \alpha \geq 0) \\
&= \max_{x_{j-1}} \underbrace{p(x_j \mid x_{j-1})}_{\text{given}} \underbrace{M_{j-1}(x_{j-1})}_{\text{recurse}} && \text{(definition of } M_{j-1}(x_{j-1})\text{)}
\end{aligned}
$$

- We also store the argmax over x_{j-1} for each (j, s) .
  - Once we have $M_j(x_j = s)$ for all j and s values, backtrack using these values to solve problem.

Example: Decoding the Plane of Doom

- We have $M_1(x_1) = p(x_1)$ so in "plane of doom" we have

$$M_1(\texttt{land}) = 0.4, \quad M_1(\texttt{crash}) = 0.3, \quad M_1(\texttt{explode}) = 0.3.$$

- We have $M_2(x_2) = \max_{x_1} p(x_2 \mid x_1) M_1(x_1)$ so we get

$$M_2(\texttt{alive}) = 0.4, \quad M_2(\texttt{dead}) = 0.3.$$

- $M_2(2) \neq p(x_2 = 2)$ because we needed to choose either `crash` or `explode`.
  - And notice that $\sum_{c=1}^{k} M_2(x_j = c) \neq 1$ (this is not a distribution over x_2).

- We maximize $M_2(x_2)$ to find that the optimal decoding ends with `alive`.
  - We now need to backtrack to find the state that led to `alive`, giving `land`.

Viterbi Decoding

- The Viterbi decoding dynamic programming algorithm:
  1. Set $M_1(x_1) = p(x_1)$ for all x_1.
  2. Compute $M_2(x_2)$ for all x_2, store argmax of x_1 leading to each x_2.
  3. Compute $M_3(x_3)$ for all x_3, store argmax of x_2 leading to each x_3.
  4. ...
  5. Maximize $M_d(x_d)$ to find value of x_d in a decoding.
  6. Bactrack to find the value of x_{d-1} that led to this x_d.
  7. Backtrack to find the value of x_{d-2} that led to this x_{d-1}.
  8. ...
  9. Backtrack to find the value of x_1 that led to this x_2.

- For a fixed j, computing all $M_j(x_j)$ given all $M_{j-1}(x_{j-1})$ costs $O(k^2)$.
  - Total cost is only $O(dk^2)$ to search over all k^d paths.
  - Has numerous applications, like decoding digital TV.

Viterbi Decoding

- What Viterbi decoding data structures might look like ($d = 4, k = 3$):

$$M = \begin{bmatrix} 0.25 & 0.25 & 0.50 \\ 0.35 & 0.15 & 0.05 \\ 0.10 & 0.05 & 0.05 \\ 0.02 & 0.03 & 0.05 \end{bmatrix}, \quad B = \begin{bmatrix} \emptyset & \emptyset & \emptyset \\ 1 & 1 & 3 \\ 2 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix}.$$

- The $d \times k$ matrix M stores the values $M_j(s)$, while B stores the argmax values.

- From the last row of M and the backtracking matrix B, the decoding is $x_1 = 1, x_2 = 2, x_3 = 1, x_4 = 3$.

Conditional Probabilities in Markov Chains: Easy Case

- How do we compute conditionals like $\Pr(x_j = c \mid x_{j'} = c')$ in Markov chains?

- Consider conditioning on an earlier time, like computing $p(x_{10} \mid x_3)$:
  - We are given the value of x_3.
  - We obtain $p(x_4 \mid x_3)$ by looking it up among transition probabilities.
  - We can compute $p(x_5 \mid x_3)$ by adding conditioning to the CK equations,

$$
\begin{aligned}
p(x_5 \mid x_3) &= \sum_{x_4} p(x_5, x_4 \mid x_3) && \text{(marginalizing)} \\
&= \sum_{x_4} p(x_5 \mid x_4, x_3) p(x_4 \mid x_3) && \text{(product rule)} \\
&= \sum_{x_4} \underbrace{p(x_5 \mid x_4)}_{\text{given}} \underbrace{p(x_4 \mid x_3)}_{\text{recurse}} && \text{(Markov property).}
\end{aligned}
$$

  - Repeat this to find $p(x_6 \mid x_3)$, then $p(x_7 \mid x_3)$, up to $p(x_{10} \mid x_3)$.

Conditional Probabilities in Markov Chains with "Forward" Messages

- How do we condition on a future time, like computing $p(x_3 \mid x_6)$?
  - Need to sum over "past" values x_1 and x_2, and over "future" values x_4 and x_5.

$$
\begin{aligned}
p(x_3 \mid x_6) \propto p(x_3, x_6) &= \sum_{x_5}\sum_{x_4}\sum_{x_2}\sum_{x_1} p(x_1, x_2, x_3, x_4, x_5, x_6) \\
&= \sum_{x_5}\sum_{x_4}\sum_{x_2}\sum_{x_1} p(x_6 \mid x_5)p(x_5 \mid x_4)p(x_4 \mid x_3)p(x_3 \mid x_2)p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x_5} p(x_6 \mid x_5) \sum_{x_4} p(x_5 \mid x_4)p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x_5} p(x_6 \mid x_5) \sum_{x_4} p(x_5 \mid x_4)p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) M_2(x_2) \\
&= \sum_{x_5} p(x_6 \mid x_5) \sum_{x_4} p(x_5 \mid x_4)p(x_4 \mid x_3) M_3(x_3) \\
&= \sum_{x_5} p(x_6 \mid x_5) M_5(x_5) = M_6(x_6)
\end{aligned}
$$

- The forward message $M_j(x_j)$ gives "everything you need to know up to time j, for this x_j value."
- Value of M_6 depends on x_3 (for $j > 3$); to get $p(x_3 \mid x_6)$, normalize by sum for all x_3.

Conditional Probabilities in Markov Chains with "Backward" Messages

- We could exchange order of sums to do computation "backwards" in time:

$$
\begin{aligned}
p(x_3 \mid x_6) &= \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)p(x_5 \mid x_4)p(x_6 \mid x_5) \\
&= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1)p(x_3 \mid x_2) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \\
&= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1)p(x_3 \mid x_2) \sum_{x_4} p(x_4 \mid x_3)V_4(x_4) \\
&= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1)p(x_3 \mid x_2)V_3(x_3) \\
&= \sum_{x_1} p(x_1)V_1(x_1)
\end{aligned}
$$

- The V_j summarize "everything you need to know after time j for this x_j value".
  - Sometimes called "cost to go" function, as in "what is the cost for going to x_j."
  - Sometimes called a value function, as in "what is the future value of being in x_j."

Motivation for Forward-Backward Algorithm

- Why do care about being able to solve this "forward" or "backward" in time?
  - Cost is $O(dk^2)$ in both directions to compute conditionals in Markov chains.
- Consider computing $p(x_1 \mid A)$, $p(x_2 \mid A)$,..., $p(x_d \mid A)$ for some event A.
  - Need all these conditionals to add features, compute conditionals with neural networks, or partial observations (as in hidden Markov models, HMMs).
- We could solve this in $O(dk^2)$ for each time, giving a total cost of $O(d^2k^2)$.
  - Using forward messages $M_j(x_j)$ at each time, or backwards messages $V_j(x_j)$.
- Alternately, the forward-backward algorithm computes all conditionals in $O(dk^2)$.
  - By doing one "forward" pass and one "backward" pass with appropriate messages.

Potential Function Representation of Markov Chains

- Forward-backward algorithm considers probabilities written in the form

$$p(x_1, x_2, \ldots, x_d) = \frac{1}{Z}\left(\prod_{j=1}^{d} \phi_j(x_j)\right)\left(\prod_{j=2}^{d} \psi_j(x_j, x_{j-1})\right).$$

- The ϕ_j and ψ_j functions are called potential functions.
  - They can map from a state (ϕ) or two states (ψ) to a non-negative number.
  - Normalizing constant Z ensures we sum/integrate to 1 (over all x_1, x_2,...,x_d).

- We can write Markov chains in this form by using (in this case $Z = 1$):
  - $\phi_1(x_1) = p(x_1)$ and $\phi_j(x_j) = 1$ when $j \neq 1$.
  - $\psi_j(x_{j-1}, x_j) = p(x_j \mid x_{j-1})$.

- Why do we need the ϕ_j functions?
  - To condition on $x_j = c$, set $\phi_j(c) = 1$ and $\phi_j(c') = 0$ for $c' \neq c$.
  - For "hidden Markov models" (HMMs), the ϕ_j will be the "emission probabilities".
  - For neural networks, ϕ_j will be $\exp$(neural network output) (generalizes softmax).

Forward-Backward Algorithm

- Forward pass in forward-backward algorithm (generalizes CK equations):
  - Set each $M_1(x_1) = \phi_1(x_1)$.
  - For $j = 2$ to $j = d$, set each $M_j(x_j) = \sum_{x_{j-1}} \phi_j(x_j)\psi_j(x_j, x_{j-1})M_{j-1}(x_{j-1})$.
    - "Multiply by new terms at time j, summing up over x_{j-1} values."

- Backward pass in forward-backward algorithm:
  - Set each $V_d(x_d) = \phi_d(x_d)$.
  - For $(d-1)$ to $j = 1$, set each $V_j(x_j) = \sum_{x_{j+1}} \phi_j(x_j)\psi_{j+1}(x_{j+1}, x_j)V_{j+1}(x_{j+1})$.

- We then have that $p(x_j) \propto \frac{M_j(x_j)V_j(x_j)}{\phi_j(x_j)}$.
  - Not obvious; see bonus for how it gives conditional in Markov chain.
  - We divide by $\phi_j(x_j)$ since it is included in both the forward and backward messages.
    - You can alternately shift ϕ_j to earlier/later message to remove division.
- We can also get the normalizing constant as $Z = \sum_{c=1}^{k} M_d(c)$.

Sequential Monte Carlo (Particle Filters)

bonus!

- For continuous non-Gaussian Markov chains, we usually need approximate inference.
- A popular strategy in this setting is sequential Monte Carlo (SMC).
  - Importance sampling where proposal q_t changes over time from simple to posterior.
  - AKA sequential importance sampling, annealed importance sampling, particle filter.
    - And can be viewed as a special case of genetic algorithms.
  - "Particle Filter Explained without Equations": `https://www.youtube.com/watch?v=aUkBa1zMKv4`

Forward-Backward for Decoding and Sampling

bonus!

- Viterbi decoding can be generalized to use potentials ϕ and ψ:
  - Compute forward messages, but with summation replaced by maximization:

$$M_j(x_j) \propto \max_{x_{j-1}} \phi_j(x_j)\psi_j(x_j, x_{j-1})M_{j-1}(x_{j-1}).$$

  - Find the largest value of $M_d(x_d)$, then backtrack to find decoding.

- Forward-filter backward-sample is a potentials (ϕ and ψ) variant for sampling.
  - Forward pass is the same.
  - Backward pass generates samples (ancestral sampling backwards in time):
    - Sample x_d from $M_d(x_d) = p(x_d)$.
    - Sample x_{d-1} using $M_{d-1}(x_{d-1})$ and sampled x_d.
    - Sample x_{d-2} using $M_{d-2}(x_{d-2})$ and sampled x_{d-1}.
    - (continue until you have sampled x_1)

Outline

Markov Chains for Monte Carlo Estimation

- We've been discussing inference in Markov chains.
  - Sampling, marginals, stationary distribution, decoding, conditionals.

- We can also use Markov chains for inference in other models.
  - Most common way to do this is Markov chain Monte Carlo (MCMC).
  - Widely used for approximate inference, e.g. in Bayesian logistic regression.

- High-level idea of MCMC:
  - We want to use Monte Carlo estimates with a distribution p.
    - But we don't know how to generate IID samples from p.
  - Design a homogeneous Markov chain whose stationary distribution is p.
    - This is usually surprisingly easy to do.
  - Use ancestral sampling to sample from a long version of this Markov chain.
  - Use the Markov chain samples within the Monte Carlo approximation.

Degenerate Example: "Pointless MCMC"

- Consider finding the expected value of a fair die:
  - For a 6-sided die, the expected value is 3.5.

- Consider the following "pointless MCMC" algorithm:
  - Start with some initial value, like "4".
  - At each step, roll the die and generate a random number u:
    - If $u < 0.5$, "accept" the roll and take the roll as the next sample.
    - Otherwise, "reject" the roll and take the old value (e.g. "4") as the next sample.

- Generates samples from a Markov chain with this transition probability:

$$q(x_t \mid x_{t-1}) = \begin{cases} 7/12 & x_t = x_{t-1} \\ 1/12 & x_t \neq x_{t-1} \end{cases}.$$

  - Using q to avoid confusion with the probability p we want to sample.

Degenerate Example: "Pointless MCMC"

- Pointless MCMC in action:
  - Start with "4", so record "4".
  - Roll a "6" and generate 0.234, so record "6".
  - Roll a "3" and generate 0.612, so record "6".
  - Roll a "2" and generate 0.523, so record "6".
  - Roll a "3" and generate 0.125, so record "3".
  - Roll a "2" and generate 0.433, so record "2".

- So our samples are 4,6,6,6,3,2...
  - If you run this long enough, you will spend $1/6$ of the time on each number.
    - Stationary distribution of pointless MCMC is $\pi(c) = 1/6$, so

$$\pi(x) = p(x),$$

      which is the key feature underlying MCMC methods.
  - This property lets us use the dependent samples within Monte Carlo.

- It is "pointless" since it assumes we can generate IID samples from p.
  - If you can do that, don't use MCMC to get approximate samples!

Markov Chain Monte Carlo (MCMC)

- Markov chain Monte Carlo (MCMC):
  - Design a Markov chain that has $\pi(x) = p(x)$.
    - For large enough k, a sample x^k from the chain will be distributed according to $p(x)$.
    - We changed notation a bit: x^1 is the first sampled state, x^2 the second, $\ldots$, x^n last.
  - Use the Markov chain samples within a Monte Carlo estimator,

$$\mathbb{E}[g(x)] \approx \frac{1}{n} \sum_{t=1}^{n} g(x^i).$$

- Law of large numbers can be generalized to show this converges as $n \to \infty$.
  - "Ergodic theorem."
  - But convergence is slower since we're generating dependent samples.

- A popular way to design the Markov chain is Metropolis-Hastings algorithm.
  - Oldest algorithm out of the "10 Best Algorithms of the 20th Century".

Special Case: Metropolis Algorithm

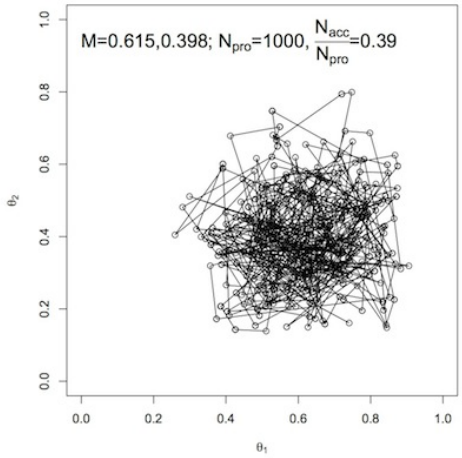
- The Metropolis algorithm for sampling from a continuous target $p(x)$:
  - Assumes we can evaluate p up to a normalizing constant, $p(x) = \tilde{p}(x)/Z$.
  - Start with some initial value x^0.
  - On each iteration add zero-mean Gaussian noise to x^{t-1} to give proposal $\hat{x}^t$.
    - And generate a u uniformly between 0 and 1.
  - "Accept" the proposal and set $x^t = \hat{x}^t$ if

$$u \leq \frac{\tilde{p}(\hat{x}^t)}{\tilde{p}(x^{t-1})}, \quad \frac{\text{(probability of proposed)}}{\text{(probability of current)}}$$

  - Otherwise "reject" the sample and use x^{t-1} again as the next sample x^t.
    - Proposals that increase probability are always accepted.
    - Proposals that decrease probability might be accepted or rejected.

  - A random walk, but sometimes rejecting steps that decrease probability:
    - A valid MCMC algorithm on continuous densities, but convergence may be slow.
    - You can implement this even if you don't know normalizing constant.

Metropolis Algorithm in Action




```
while True:
  xhat = x + \
    rs.multivariate_normal(cov=Sigma)
  u = rs.random()
  if u < p(xhat) / p(x):
      x = xhat
  yield x
```

http://www.columbia.edu/~cjd11/charles_dimaggio/DIRE/styled-4/

Metropolis Algorithm Analysis

- Markov chain with transitions $q_{s \to s'} = q(x^t = s' \mid x^{t-1} = s)$ is reversible if

$$\pi(s) q_{s \to s'} = \pi(s') q_{s' \to s},$$

  for some distribution π (this condition is called detailed balance).

- Reversibility implies π is a stationary distribution:

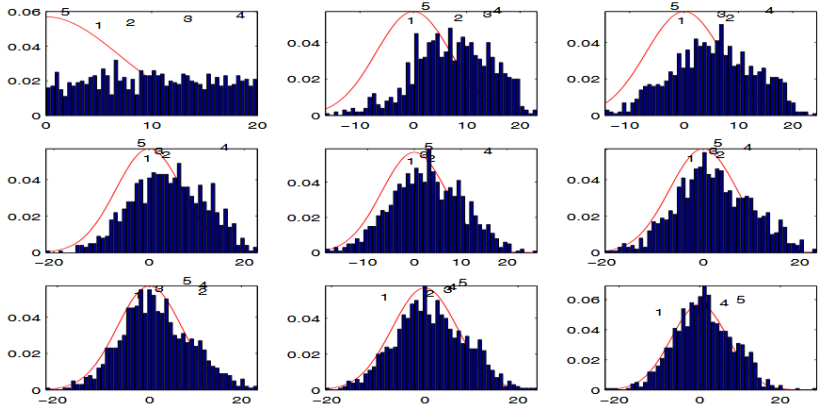
$$\begin{aligned} \pi^+(s) = \sum_{s'} \pi(s') q_{s' \to s} &= \sum_{s'} \pi(s) q_{s \to s'} && \text{(detailed balance for each term)} \\ &= \pi(s) \underbrace{\sum_{s'} q_{s \to s'}}_{1} \\ &= \pi(s) && \text{(stationary condition).} \end{aligned}$$

- Metropolis is reversible with $\pi = p$ (bonus slide), so p is stationary distribution.
  - And positive transition probabilities mean π exists, and is unique/reached.

Markov Chain Monte Carlo

MCMC sampling from a Gaussian:

From top left to bottom right: histograms of 1000 independent Markov chains with a normal distribution as target distribution.



http://www.cs.ubc.ca/~arnaud/stat535/slides10.pdf

MCMC Implementation Issues

- In practice, we often don't take all samples in our Monte Carlo estimate:
  - Burn in: throw away the initial samples when we're far from stationary.
  - Thinning: only keep every k samples, since they'll be highly correlated.

- Two common ways that MCMC is applied:
  1. Sample from a huge number of Markov chains for a long time, use final states.
     - Great for parallelization.
     - No need for thinning, since you throw all but last samples.
     - Need to worry about burn in for each chain.
  2. Sample from one Markov chain for a really long time, use states across time.
     - Less worry about burn in.
     - May need to worry about thinning.

- It can very hard to diagnose if we have reached stationary distribution.
  - It's PSPACE-hard – even harder than NP-hard.
  - Various heuristics exist.

Summary

- Viterbi decoding allow efficient decoding with Markov chains.
  - A special case of dynamic programming.
- Potential representation of Markov chains (more general formulation).
  - Non-negative potential ϕ at each time and ψ for each transition.
- Forward-backward generalizes CK equations for potentials.
  - Allows computing all marginals in $O(dk^2)$.
- Markov chain Monte Carlo (MCMC) approximates complicated expectations.
  - Generate samples from a Markov chain that has p as stationary distribution.
  - Use these samples within a Monte Carlo approximation.

- Next time: lots more MCMC and lots of DAGs.

Computing Markov Chain Conditional using Forward-Backward

bonus!

$$
\begin{aligned}
p(x_3 \mid x_6) &\propto \sum_{x_4}\sum_{x_5}\sum_{x_2}\sum_{x_1} p(x_1, x_2, x_3, x_4, x_5, x_6) \quad \text{(set up both sums to work "outside in")} \\
&= \sum_{x_4}\sum_{x_5}\sum_{x_2}\sum_{x_1} p(x_4 \mid x_3)p(x_5 \mid x_4)p(x_6 \mid x_5)p(x_3 \mid x_2)p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1)M_1(x_1) \\
&= \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \sum_{x_2} p(x_3 \mid x_2)M_2(x_2) \\
&= \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5)M_3(x_3) \\
&= M_3(x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \quad \text{(take } M_3(x_3) \text{ outside sums)} \\
&= M_3(x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5)V_6(x_6) \quad (V_6(x_6) = 1) \\
&= M_3(x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)V_5(x_5) \\
&= M_3(x_3) \sum_{x_4} p(x_4 \mid x_3)V_4(x_4) \\
&= M_3(x_3)V_3(x_3) \quad (\phi_3(x_3) = 1 \text{ so no division, normalize over } x_3 \text{ values to get final answer})
\end{aligned}
$$

Metropolis Algorithm Analysis

bonus!

- Metropolis algorithm has $q_{s \to s'} > 0$ (sufficient to guarantee stationary distribution is unique and we reach it), and satisfies detailed balance with target distribution p,

$$p(s) q_{s \to s'} = p(s') q_{s' \to s}.$$

- We can show this by defining the transition probabilities as

$$c_{s-s'} = \frac{\exp\left(-\frac{1}{2}(s-s')\Sigma^{-1}(s-s')\right)}{(2\pi \det \Sigma)^{d/2}} \qquad q_{s \to s'} = c_{s-s'} \min\left\{1, \frac{\tilde{p}(s')}{\tilde{p}(s)}\right\},$$

and observing that

$$\begin{aligned} p(s) q_{s \to s'} &= c_{s-s'} p(s) \min\left\{1, \frac{\tilde{p}(s')}{\tilde{p}(s)}\right\} = c_{s-s'} p(s) \min\left\{1, \frac{\frac{1}{Z}\tilde{p}(s')}{\frac{1}{Z}\tilde{p}(s)}\right\} \\ &= c_{s-s'} p(s) \min\left\{1, \frac{p(s')}{p(s)}\right\} = c_{s-s'} \min\left\{p(s), p(s')\right\} \\ &= p(s') c_{s'-s} \min\left\{1, \frac{p(s)}{p(s')}\right\} = p(s') q_{s' \to s}. \end{aligned}$$