Introduction to Inference in Trees

Prof. Dr. H. H. Takada

Quantitative Research – Itaú Asset Management
Institute of Mathematics and Statistics – University of São Paulo

Inference

Inference corresponds to using the distribution to answer questions about the environment.

---

examples

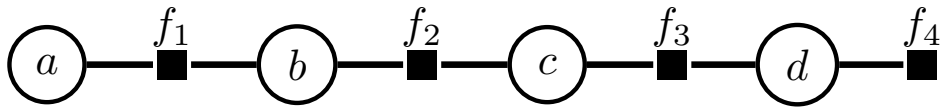
- What is the probability $p(x = 4|y = 1, z = 2)$?
- What is the most likely joint state of the distribution $p(x, y)$?
- What is the entropy of the distribution $p(x, y, z)$?
- What is the probability that this example is in class 1?
- What is the probability the stock market will do down tomorrow?

Computational Efficiency

- Inference can be computationally very expensive and we wish to characterize situations in which inferences can be computed efficiently.
- For singly-connected graphical models, and certain inference questions, there (usually) exist efficient algorithms based on the concept of message passing.
- In general, the case of multiply-connected models is computationally inefficient.

Sum-Product Algorithm - Non-Branching Tree

$p(a,b,c,d) \propto f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d)$ $\quad a,b,c,d$ binary variables

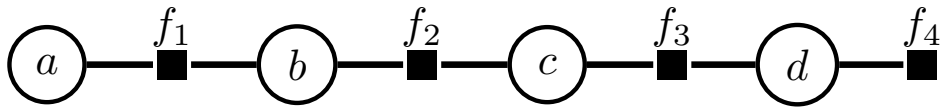


$$p(a) = \sum_{b,c,d} p(a,b,c,d)$$

$$\propto \sum_{b,c,d} f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d) \Rightarrow 2^3 \text{ sums}$$

Sum-Product Algorithm - Non-Branching Tree

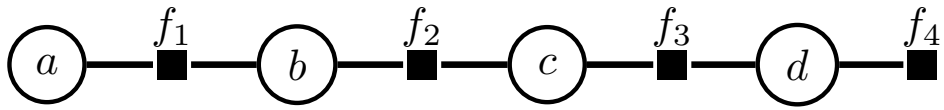
$p(a,b,c,d) \propto f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d)$ $\quad a,b,c,d$ binary variables



$$
\begin{aligned}
p(a) &= \sum_{b,c,d} p(a,b,c,d) \\
&\propto \sum_{b,c,d} f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d) \Rightarrow 2^3 \text{ sums} \\
&= \sum_{b} f_1(a,b) \sum_{c} f_2(b,c) \sum_{d} f_3(c,d)\, f_4(d) \Rightarrow 2 \times 3 \text{ sums}
\end{aligned}
$$

Sum-Product Algorithm - Non-Branching Tree

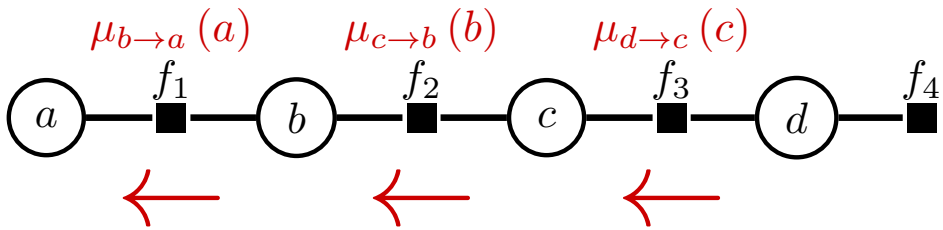
$p(a,b,c,d) \propto f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d)$ $\quad a,b,c,d$ binary variables



$$
\begin{aligned}
p(a) &= \sum_{b,c,d} p(a,b,c,d) \\
&\propto \sum_{b,c,d} f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d) \\
&= \sum_{b} f_1(a,b) \underbrace{\sum_{c} f_2(b,c) \underbrace{\sum_{d} f_3(c,d)\, f_4(d)}_{\mu_{d\to c}(c)}}_{\mu_{c\to b}(b)} \\
&\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\mu_{b\to a}(a)}
\end{aligned}
$$

Sum-Product Algorithm - Non-Branching Tree

$p(a,b,c,d) \propto f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d)$ $\quad a,b,c,d$ binary variables

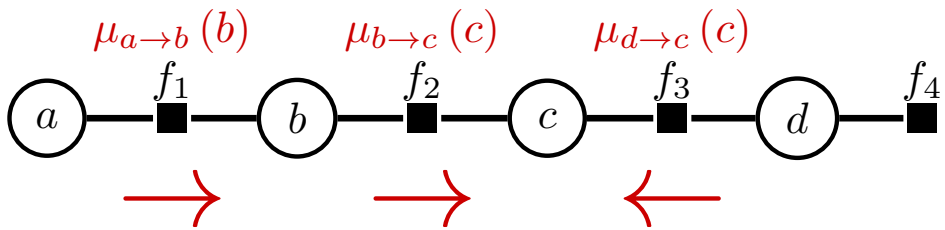


Passing variable-to-variable messages from d up to a

$$
\begin{aligned}
p(a) &= \sum_{b,c,d} p(a,b,c,d) \\
&\propto \sum_{b,c,d} f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d) \\
&= \sum_{b} f_1(a,b) \underbrace{\sum_{c} f_2(b,c) \underbrace{\sum_{d} f_3(c,d)\, f_4(d)}_{\mu_{d\to c}(c)}}_{\mu_{c\to b}(b)} \\
&\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\mu_{b\to a}(a)}
\end{aligned}
$$

Sum-Product Algorithm - Non Branching Tree

For $p(c)$ need to send messages in both directions

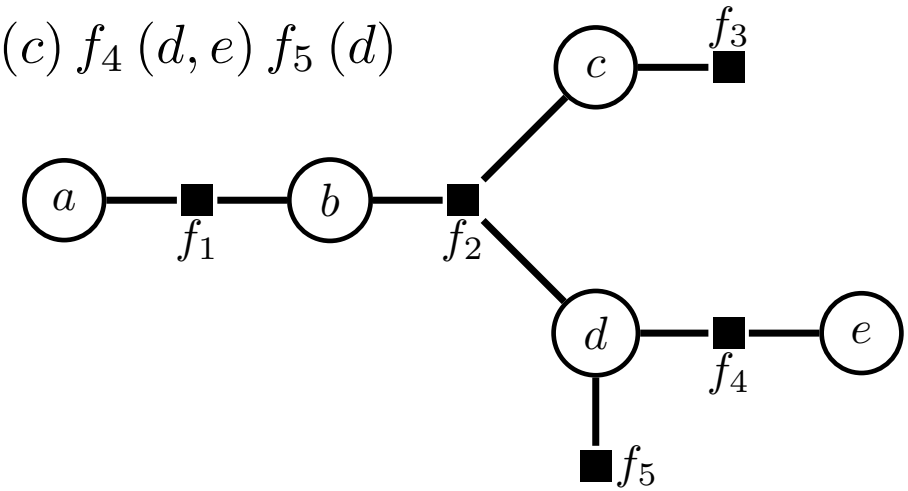


$$p(c) \propto \sum_{a,b,d} f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d)$$

$$= \underbrace{\sum_b \underbrace{\sum_a f_1(a,b)}_{\mu_{a\to b}(b)}\, f_2(b,c)}_{\mu_{b\to c}(c)} \underbrace{\sum_d f_3(c,d)\, f_4(d)}_{\mu_{d\to c}(c)}$$

Sum-Product Algorithm – Branching Tree

$$p(a,b,c,d,e) \propto f_1(a,b)\, f_2(b,c,d)\, f_3(c)\, f_4(d,e)\, f_5(d)$$



Define factor-to-variable messages and variable-to-factor messages

$$p(a) \propto \sum_b f_1(a,b) \sum_{c,d} f_2(b,c,d) \underbrace{f_3(c)}_{\mu_{c\to f_2}(c)=\mu_{f_3\to c}(c)} \underbrace{f_5(d) \underbrace{\sum_e f_4(d,e)}_{\mu_{f_4\to d}(d)}}_{\mu_{d\to f_2}(d)}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\mu_{b\to f_1}(b)=\mu_{f_2\to b}(b)}$$

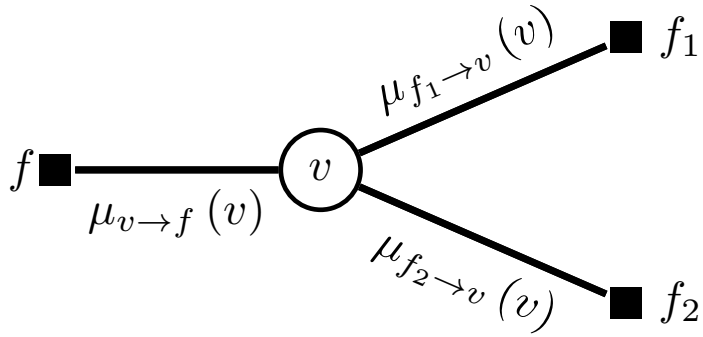
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\mu_{f_1\to a}(a)}$$

$\Rightarrow$ **Marginal inference for a singly-connected structure is 'easy'.**

Sum-Product Algorithm for Factor Graphs

Variable to factor message

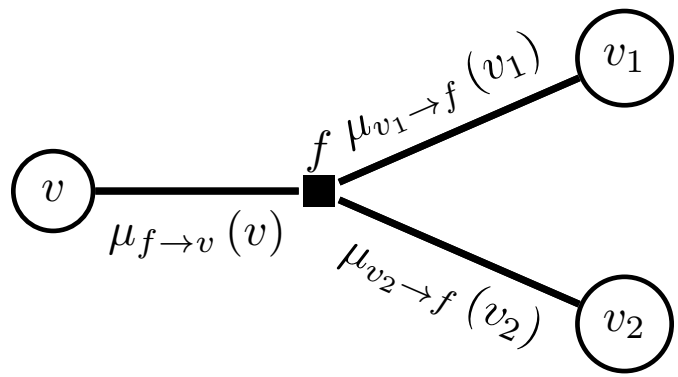
$$\mu_{v \to f}(v) = \prod_{f_i \sim v \setminus f} \mu_{f_i \to v}(v)$$



Messages from extremal variables are set to 1

Factor to variable message

$$\mu_{f \to v}(v) = \sum_{\{v_i\}} f(v, \{v_i\}) \prod_{v_i \sim f \setminus v} \mu_{v_i \to f}(v_i)$$



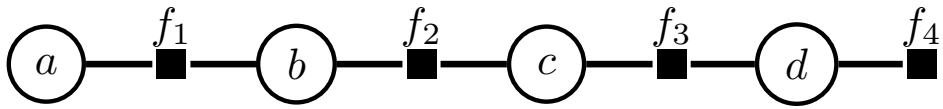
Messages from extremal factors are set to the factor

Marginal

$p(v) \propto \prod_{f_i \sim v} \mu_{f_i \to v}(v)$

Max-Product algorithm

$p(a,b,c,d) \propto f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d)$ $\quad a,b,c,d$ binary variables



$$\max_{a,b,c,d} p(a,b,c,d) = \max_{a,b,c,d} f_1(a,b)\, f_2(b,c)\, f_3(c,d)\, f_4(d)$$

$$= \max_a \max_b f_1(a,b) \max_c f_2(b,c) \underbrace{\max_d f_3(c,d)\, f_4(d)}_{\mu_{d\to c}(c)}$$

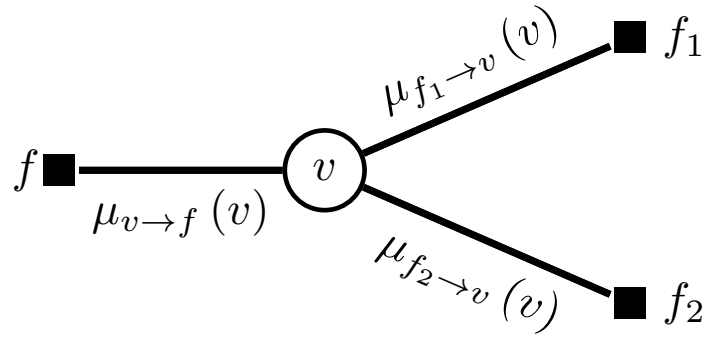
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\mu_{c\to b}(b)}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\mu_{b\to a}(a)}$$

Max-Product Algorithm for Factor Graphs

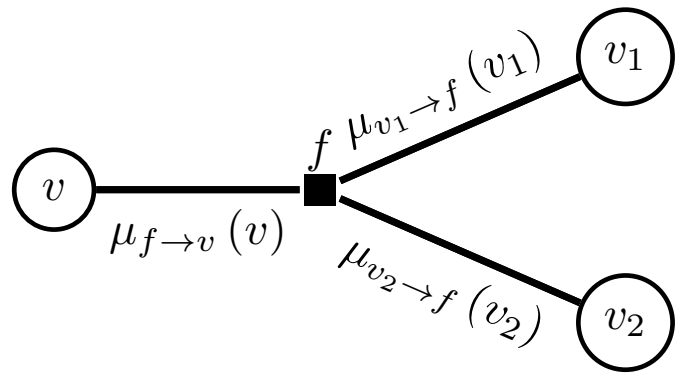
Variable to factor message

$$\mu_{v \to f}(v) = \prod_{f_i \sim v \setminus f} \mu_{f_i \to v}(v)$$



Factor to variable message

$$\mu_{f \to v}(v) = \max_{\{v_i\}} f(v, \{v_i\}) \prod_{v_i \sim f \setminus v} \mu_{v_i \to f}(v_i)$$



Most probable state (of joint)

$$v^* = \operatorname*{argmax}_v \prod_{f_i \sim v} \mu_{f_i \to v}(v)$$

Code

Matlab

```
>> setup;
>> demoSumprod; % test the Sum-Product algorithm
>> demoMaxprod; % test the Max-Product algorithm
>> demoMaxNprod; % test the Max-N-Product algorithm
```

Message Passing

- Also known as ‘belief propagation’ or ‘dynamic programming’.
- Note that for non-branching graphs (they look like ‘lines’), only variable to variable messages are required.
- For message passing to work we need to be able to distribute the operator over the factors (which means that the operator algebra is a semiring) and that the graph is singly-connected.
- Provided the above conditions hold, ‘marginal’ inference scales linearly with the number of nodes in the graph.

Message Passing

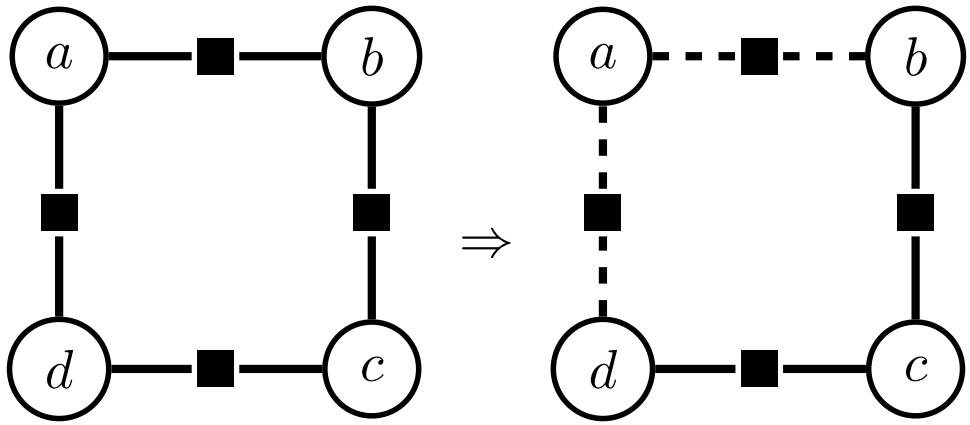
- If the graph is multiply-connected, message passing can still be implemented since it is a local algorithm. This is a popular approximation technique.
- Sometimes it is possible to identify a singly-connected structure from a multiply-connected structure by conditioning on a small set of variables (the cut-set). One can then run a set of message-passing algorithms, one for each state of the cut-set.
- What if the operator algebra is not a semiring? Won't work in general. An example is where we want

$$\max_{d,e,f} \sum_{a,b,c} p(a,b,c,d,e,f)$$

In this case, the $\max \sum$ operator is not distributive (the max of a sum is not the same as the sum of a max).

Cut-set conditioning

Identify a set of variables to reveal a set of singly-connected structures.

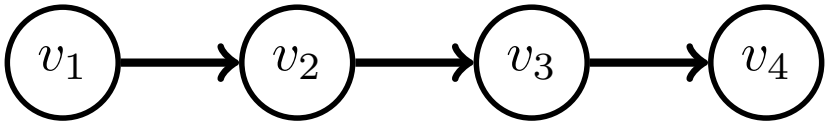


$$\max_{a,b,c,d} \phi(a,b)\phi(b,c)\phi(c,d)\phi(a,d)$$

$$= \max_{a} \underbrace{\max_{b,c,d} \phi(a,b)\phi(b,c)\phi(c,d)\phi(a,d)}_{\text{singly-connected for fixed } a}$$

Each state of the cut-set identifies a singly-connected structure, for which the inference is performed efficiently. We then have to carry out the final operations over all states of the cut-set.

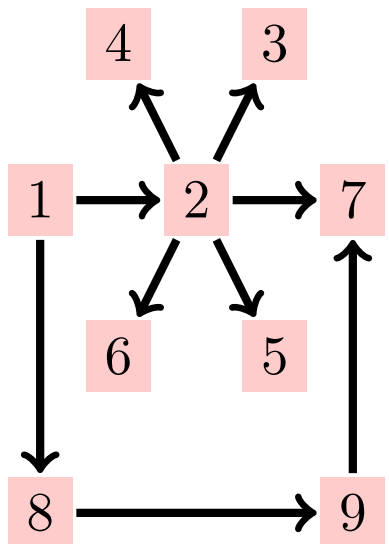
Markov Chains



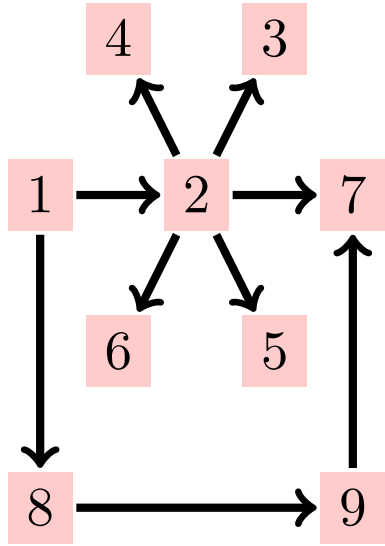
$$p(v_1, \ldots, v_T) = \underbrace{p(v_1)}_{\text{initial}} \prod_{t=2}^{T} \underbrace{p(v_t|v_{t-1})}_{\text{Transition}}$$

'Marginal' inference can be carried out in $O(T)$.
Can use a state-transition diagram to represent $p(v_t|v_{t-1})$?

Example



Most probable and shortest paths



- The shortest (unweighted) path from state 1 to state 7 is $1-2-7$.
- The most probable path from state 1 to state 7 is $1-8-9-7$ (assuming uniform transition probabilities). The latter path is longer but more probable since for the path $1-2-7$, the probability of exiting state 2 into state 7 is $1/5$.

Matlab

```
>> setup; >> demoMostProbablePath;
```

Most probable path: message passing

Supose you want to find the most probable path from state a to state b. First assume there exists a length T path, $\mathsf{a}, s_2, \ldots, s_{T-1}, \mathsf{b}$ and define the maximal path probability:

$$\begin{aligned} E(\mathsf{a} \to \mathsf{b}, T) &= \max_{s_2,\ldots,s_{T-1}} p(s_2|s_1 = \mathsf{a}) p(s_3|s_2) p(s_4|s_3) \ldots p(s_T = \mathsf{b}|s_{T-1}) \\ &= \max_{s_3,\ldots,s_{T-1}} \underbrace{\max_{s_2} p(s_2|s_1 = \mathsf{a}) p(s_3|s_2)}_{\gamma_{2\to 3}(s_3)} p(s_4|s_3) \ldots p(s_T = \mathsf{b}|s_{T-1}) \end{aligned}$$

To compute this efficiently we define messages

$$\gamma_{t\to t+1}(s_{t+1}) = \max_{s_t} \gamma_{t-1\to t}(s_t)\, p(s_{t+1}|s_t), \quad t \geq 2, \quad \gamma_{1\to 2}(s_2) = p(s_2|s_1 = \mathsf{a})$$

until the point

$$E(\mathsf{a} \to \mathsf{b}, T) = \max_{s_{T-1}} \gamma_{T-2\to T-1}(s_{T-1})\, p(s_T = \mathsf{b}|s_{T-1}) = \gamma_{T-1\to T}(s_T = \mathsf{b})$$

Now find the maximal path probability for timestep $T+1$. Since the messages up to time $T-1$ will be the same as before, we only need to compute one additional message, $\gamma_{T-1\to T}\left(s_T\right)$, from which

$$E\left(\mathsf{a}\to\mathsf{b}, T+1\right) = \max_{s_T} \gamma_{T-1\to T}\left(s_T\right) p(s_{T+1}=\mathsf{b}|s_T) = \gamma_{T\to T+1}\left(s_{T+1}=\mathsf{b}\right)$$

Proceed until we reach $E\left(\mathsf{a}\to\mathsf{b}, N\right)$ where N is the number of nodes in the graph. The optimal time t^* is then given by which of $E\left(\mathsf{a}\to\mathsf{b}, 2\right), \ldots, E\left(\mathsf{a}\to\mathsf{b}, N\right)$ is maximal. Given t^* one can begin to backtrack. Since

$$E\left(\mathsf{a}\to\mathsf{b}, t^*\right) = \max_{s_{t^*-1}} \gamma_{t^*-2\to t^*-1}\left(s_{t^*-1}\right) p(s_{t^*}=\mathsf{b}|s_{t^*-1})$$

we know the optimal state

$$\mathsf{s}^*_{t^*-1} = \operatorname*{argmax}_{s_{t^*-1}} \gamma_{t^*-2\to t^*-1}\left(s_{t^*-1}\right) p(s_{t^*}=\mathsf{b}|s_{t^*-1})$$

We can then continue to backtrack:

$$\mathsf{s}^*_{t^*-2} = \operatorname*{argmax}_{\mathsf{s}_{t^*-2}} \gamma_{t^*-3\to t^*-2}\left(s_{t^*-2}\right) p(\mathsf{s}^*_{t^*-1}|s_{t^*-2})$$

and so on.

Most probable path: message passing

Numerical issues

As it stands, the algorithm is numerically impractical since the messages are recursively multiplied by values usually less than 1 (at least for the case of probabilities). One will therefore quickly run into numerical underflow (or possibly overflow in the case of non-probabilities) with this method. This can be remedied by working in $\log$ space, and defining a form of max-sum algorithm.

Shortest weighted path

Each edge has a weight $u(s_t|s_{t-1})$. A weighted path from a to b is the sum of the weights along the path.

To solve the shortest weighted path problem we set

$$p(s_t|s_{t-1}) \propto \exp(-u(s_t|s_{t-1}))$$

where $u(s_t|s_{t-1})$ is the edge weight and is infinite if there is no edge from s_{t-1} to s_t. This method is more general than Dijkstra's algorithm which requires weights to be positive. See `demoShortestPath.m`.

Equilibrium distribution

It is interesting to know how the marginal $p(x_t)$ evolves through time:

$$p(x_t = i) = \sum_j \underbrace{p(x_t = i|x_{t-1} = j)}_{M_{ij}} p(x_{t-1} = j)$$

The marginal $p(x_t = i)$ has the interpretation of the frequency that we visit state i at time t, given we started from $p(x_1)$ and randomly drew samples from the transition $p(x_\tau|x_{\tau-1})$. As we repeatedly sample a new state from the chain, the distribution at time t, for an initial distribution $\mathbf{p}_1(i)$ is

$$\mathbf{p}_t = \mathbf{M}^{t-1}\mathbf{p}_1$$

If, for $t \to \infty$, $\mathbf{p}_\infty$ is independent of the initial distribution $\mathbf{p}_1$, then $\mathbf{p}_\infty$ is called the equilibrium distribution of the chain.

$$p_\infty(i) = \sum_j p(x_t = i|x_{t-1} = j)p_\infty(j)$$

In matrix notation this can be written as the vector equation

$$\mathbf{p}_\infty = \mathbf{M}\mathbf{p}_\infty$$

so that the stationary distribution is proportional to the eigenvector with unit eigenvalue of the transition matrix.

Example: Page Rank

Define the matrix

$$A_{ij} = \begin{cases} 1 & \text{if website } j \text{ has a hyperlink to website } i \\ 0 & \text{otherwise} \end{cases}$$

From this we can define a Markov transition matrix with elements

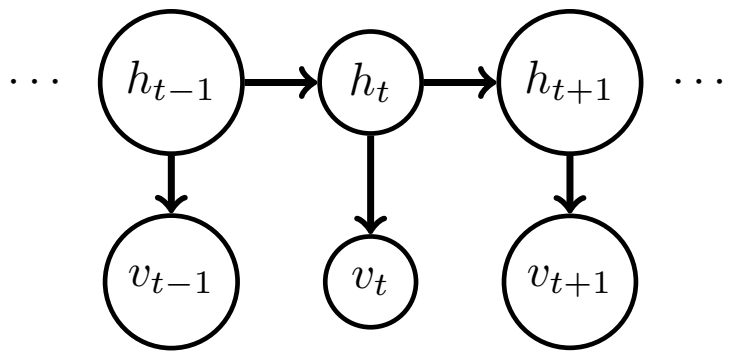
$$M_{ij} = \frac{A_{ij}}{\sum_{i'} A_{i'j}}$$

- If we jump from website to website, the equilibrium distribution component $p_\infty(i)$ is the relative number of times we will visit website i. This has a natural interpretation as the 'importance' of website i.
- For each website i a list of words associated with that website is collected. After doing this for all websites, one can make an 'inverse' list of which websites contain word w. When a user searches for word w, the list of websites that contain word is then returned, ranked according to the importance of the site.

Hidden Markov Models (HMM)

This is a popular time series model used throughout many different fields (Machine Learning, Statistics, Tracking, Bioinformatics and many many more).

- A set of discrete or continuous variables $v_1, \ldots, v_T \equiv v_{1:T}$ which represent the observed time-series.
- A set of discrete hidden variables $h_{1:T}$ that generate the observations.



$$p(v_{1:T}, h_{1:T}) = p(v_1|h_1)p(h_1)\prod_{t=2}^{T} p(v_t|h_t)p(h_t|h_{t-1})$$

$p(h_t = j|h_{t-1} = i) = \pi_{ji}$, π: transition matrix

$p(v_t = j|h_t = i) = \rho_{ji}$, ρ: emission matrix

HMM: Common inference problems

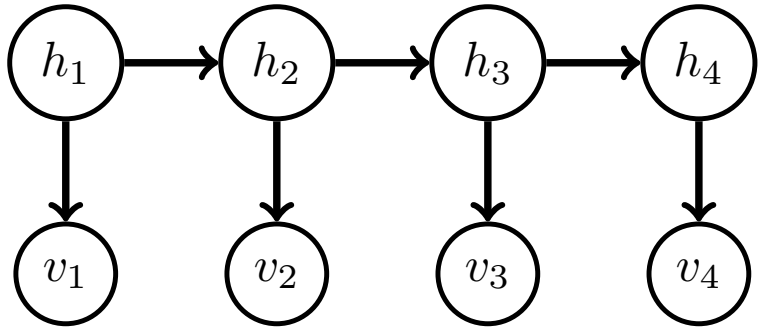
Filtering Infer h_t from $p(h_t|v_{1:t})$ which uses the observations up to time t

Smoothing Infer h_t from $p(h_t|v_{1:T})$ which also uses future observations

Viterbi Infer the most likely hidden sequence $h_{1:T}$ from $\operatorname{argmax}_{h_{1:T}} p(h_{1:T}|v_{1:T})$

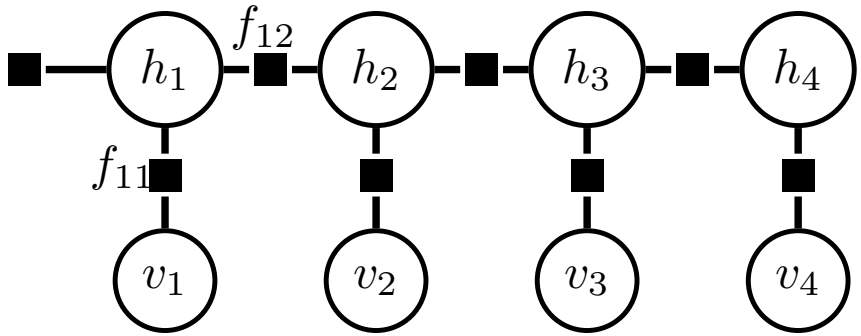
Inference in Hidden Markov Models

Belief network representation of a HMM:



As a factor graph:

$f_{11} = p(v_1|h_1)$
$f_{12} = p(h_2|h_1)$



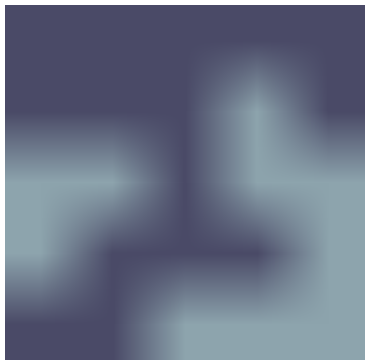
- Filtering: carried our by passing messages up and to the right.
- Smoothing: combine filtering messages with messages up and to the left. Viterbi computed similarly.

Localisation example – Part I

Problem: You're asleep upstairs in your house and awoken by a burglar on the ground floor. You want to figure out where the burglar might be based on a sequence of noise information.

You mentally partition the ground floor into a 5×5 grid. For each grid position

- you know the probability that if someone is in that position the floorboard will creak
- you know the probability that if someone is in that position he will bump into something in the dark
- you assume that the burglar can move only into a neighbor grid square with uniform probability

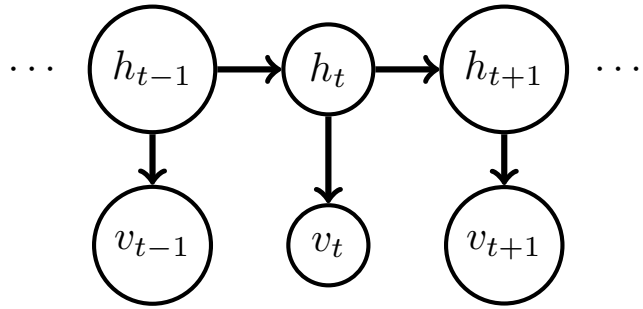


Prob. of creak

Prob. of bump

Localisation example – Part II

We can represent the scenario using a HMM where



- The hidden variable h_t represents the position of the burglar in the grid at time t

  $h_t \in \{1, \ldots, 25\}$

- The visible variable v_t represents creak/bump at time t

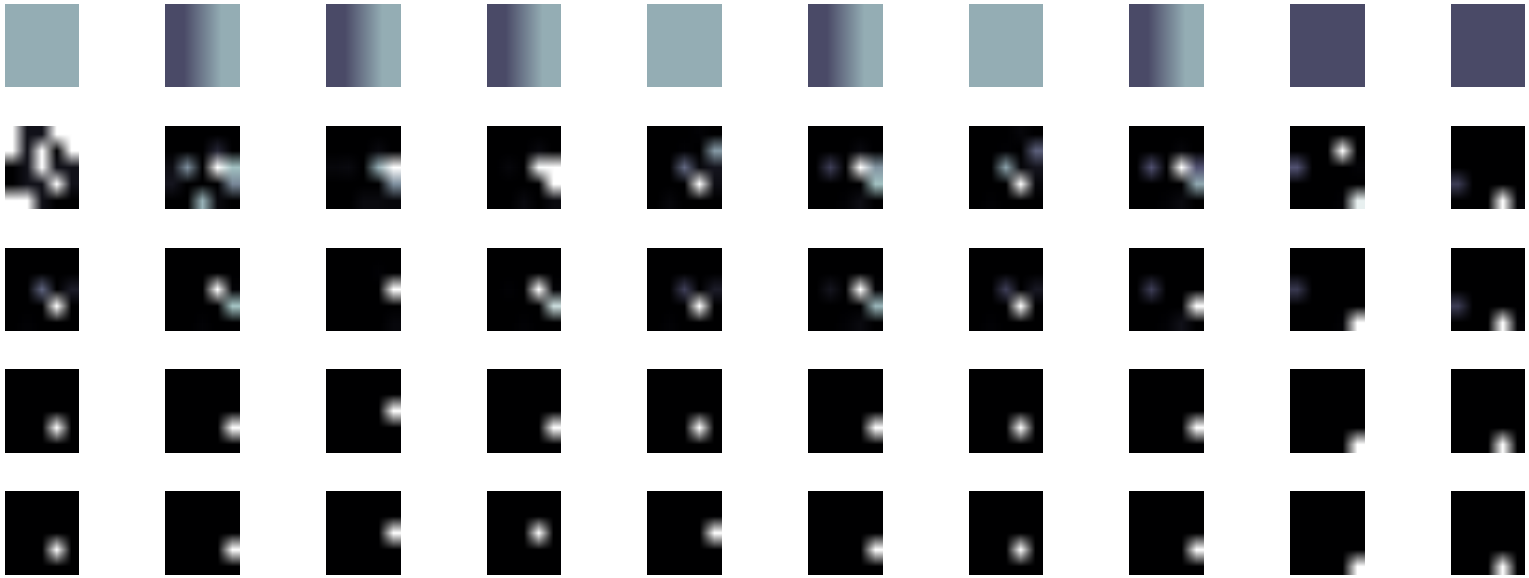
  v=1: no creak, no bump

  v=2: creak, no bump

  v=3: no creak, bump

  v=4: creak, bump

Localisation example – Part III



(top) Observed creaks and bumps for 10 time-steps

(below top) Filtering $p(h_t|v_{1:t})$

(middle) Smoothing $p(h_t|v_{1:10})$

(above bottom) Most likely sequence $\underset{h_{1:T}}{\text{argmax}}\ p(h_{1:T}|v_{1:T})$

(bottom) True Burglar position

Matlab

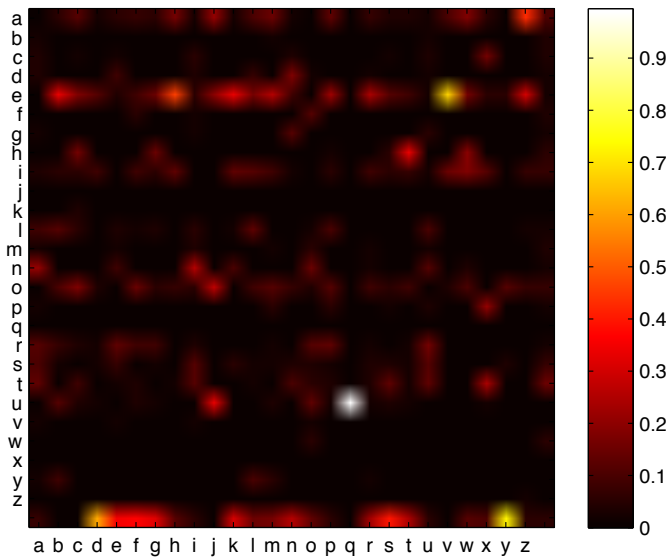
```
>> setup; demoHMMburglar;
```

Natural Language Model Example – Part I

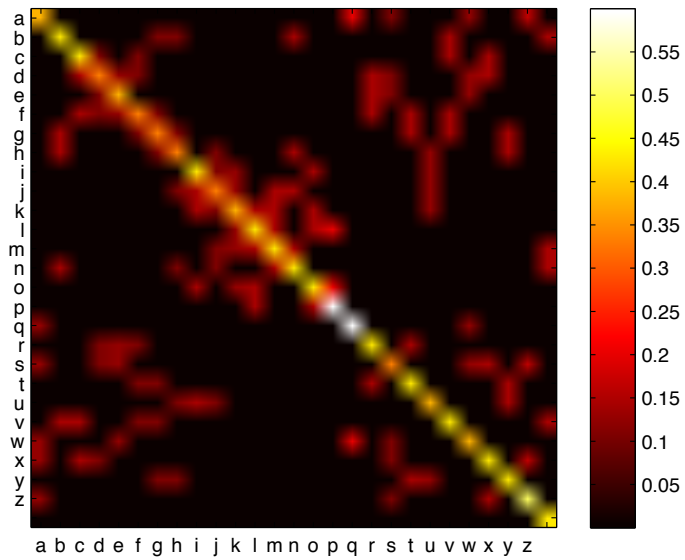
Problem: A 'stubby finger' typist has the tendency to hit either the correct key or a neighbouring key. Given a typed sequence you want to infer what is the most likely word that this corresponds to.

- The hidden variable h_t represents the intended letter at time t
- The visible variable v_t represents the letter that was actually typed at time t

We assume that there are 27 keys: lower case a to lower case z and the space bar.



Transition $p(h_t = j|h_{t-1} = i)$



Emission $p(v_t = j|h_t = i)$

Natural Language Model Example – Part II

Given the typed sequence `kezrninh` what is the most likely word that this corresponds to?

- Listing the 200 most likely hidden sequences (using a form of Viterbi)
- Discard those that are not in a standard English dictionary
- Take the most likely proper English word as the intended typed word

. . . and the answer is . . .

Matlab

```
>> setup;
>> demoHMMbigram;
```