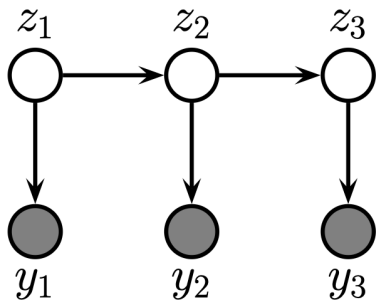
4 - Message Passing Algorithms

Belief propagation on chains

Consider the **Hidden Markov Chain model** (all the latent variables are discrete):



The corresponding joint distribution has the form:

$$p\left(\boldsymbol{y}_{1:T}, \boldsymbol{z}_{1:T}\right) = \left[p\left(\boldsymbol{z}_1\right) \prod_{t=2}^{T} p\left(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1}\right)\right] \left[\prod_{t=1}^{T} p\left(\boldsymbol{y}_t \mid \boldsymbol{z}_t\right)\right]$$

Our goal is to *infer hidden states*, i.e. to compute the *posterior marginals* $p\left(z_t \mid y_{1:T}\right)$, using Bayes rule and Markov property we have:

$$p\left(\boldsymbol{z}_t = j \mid \boldsymbol{y}_{t+1:T}, \boldsymbol{y}_{1:t}\right) \propto p\left(\boldsymbol{z}_t = j \mid \boldsymbol{y}_{1:t}\right) p\left(\boldsymbol{y}_{t+1:T} \mid \boldsymbol{z}_t = j, \cancel{\boldsymbol{y}_{1:t}}\right)$$

The *given information* are:

1. Observations $y_{1:T}$
2. Initial distribution $\pi_k = p\left(z_1 = k\right)$
3. Transition model $A_{ij} = p\left(z_t = j \mid z_{t-1} = i\right)$
4. Local evidence $p\left(y_t \mid z_t = k\right)$

Our algorithm breaks into two parts:

1. the forwards algorithm compute the filtering distribution $p\left(\boldsymbol{z}_t = j \mid \boldsymbol{y}_{1:t}\right)$ by working forward in time,
2. the backward algorithm compute the $p\left(\boldsymbol{y}_{t+1:T} \mid \boldsymbol{z}_t = j\right)$ terms by working backward in time.

The forwards algorithm (Bayes filter) (Online)

Define vectors:

- Belief states: $\alpha_t(j) \triangleq p\left(z_t = j \mid \boldsymbol{y}_{1:t}\right)$
- Local evidence: $\lambda_t(j) \triangleq p\left(\boldsymbol{y}_t \mid z_t = j\right)$
- Transition matrix: $A_{i,j} = p\left(z_t = j \mid z_{t-1} = i\right)$
  The **predict step** is given by **(Chapman-Kolmogorov equation)**:

  $$\alpha_{t|t-1}(j) \triangleq p\left(z_t = j \mid \boldsymbol{y}_{1:t-1}\right) = \sum_i p\left(z_t = j \mid z_{t-1} = i\right) p\left(z_{t-1} = i \mid \boldsymbol{y}_{1:t-1}\right) = \sum_i A_{i,j} \alpha_{t-1}($$

  The **update step** is given by

  $$\alpha_t(j) = \frac{1}{Z_t} p\left(\boldsymbol{y}_t \mid z_t = j\right) p\left(z_t = j \mid \boldsymbol{y}_{1:t-1}\right) = \frac{1}{Z_t} \lambda_t(j) \alpha_{t|t-1}(j) = \frac{1}{Z_t} \lambda_t(j) \left[\sum_i \alpha_{t-1}(i) A_{i,}\right.$$

  where we used the law of total probability, Bayes' rule, and Markov property. In matrix notation we write $\boldsymbol{\alpha}_t = \text{normalize}\left(\boldsymbol{\lambda}_t \odot \left(\mathbf{A}^\top \boldsymbol{\alpha}_{t-1}\right)\right)$. The normalization constant for each time step is given by

  $$Z_t \triangleq p\left(\boldsymbol{y}_t \mid \boldsymbol{y}_{1:t-1}\right) = \sum_{j=1}^{K} p\left(\boldsymbol{y}_t \mid \boldsymbol{z}_t = j\right) p\left(\boldsymbol{z}_t = j \mid \boldsymbol{y}_{1:t-1}\right) = \sum_{j=1}^{K} \lambda_t(j) \alpha_{t|t-1}(j)$$

The forwards-backwards algorithm (Offline)

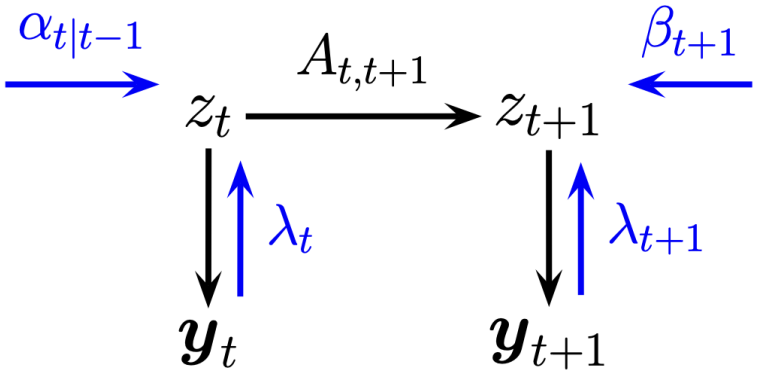
Define vectors:

- the conditional likelihood $\beta_t(j) \triangleq p\left(\boldsymbol{y}_{t+1:T} \mid \boldsymbol{z}_t = j\right)$
- then $\gamma_t(j) \triangleq p\left(\boldsymbol{z}_t = j \mid \boldsymbol{y}_{t+1:T}, \boldsymbol{y}_{1:t}\right) = \alpha_t(j)\beta_t(j) = \text{normalize}\left(\boldsymbol{\alpha}_t \odot \boldsymbol{\beta}_t\right)$.

We first have:

$$\beta_{t-1}(i) = p\left(\boldsymbol{y}_{t:T} \mid \boldsymbol{z}_{t-1} = i\right) = \sum_j \beta_t(j) \lambda_t(j) A_{i,j}$$

in matrix-vector form: $\boldsymbol{\beta}_{t-1} = \mathbf{A}\left(\boldsymbol{\lambda}_t \odot \boldsymbol{\beta}_t\right)$. The *base case* it
$\beta_T(i) = p\left(\boldsymbol{y}_{T+1:T} \mid \boldsymbol{z}_T = i\right) = p\left(\emptyset \mid \boldsymbol{z}_T = i\right) = 1$.



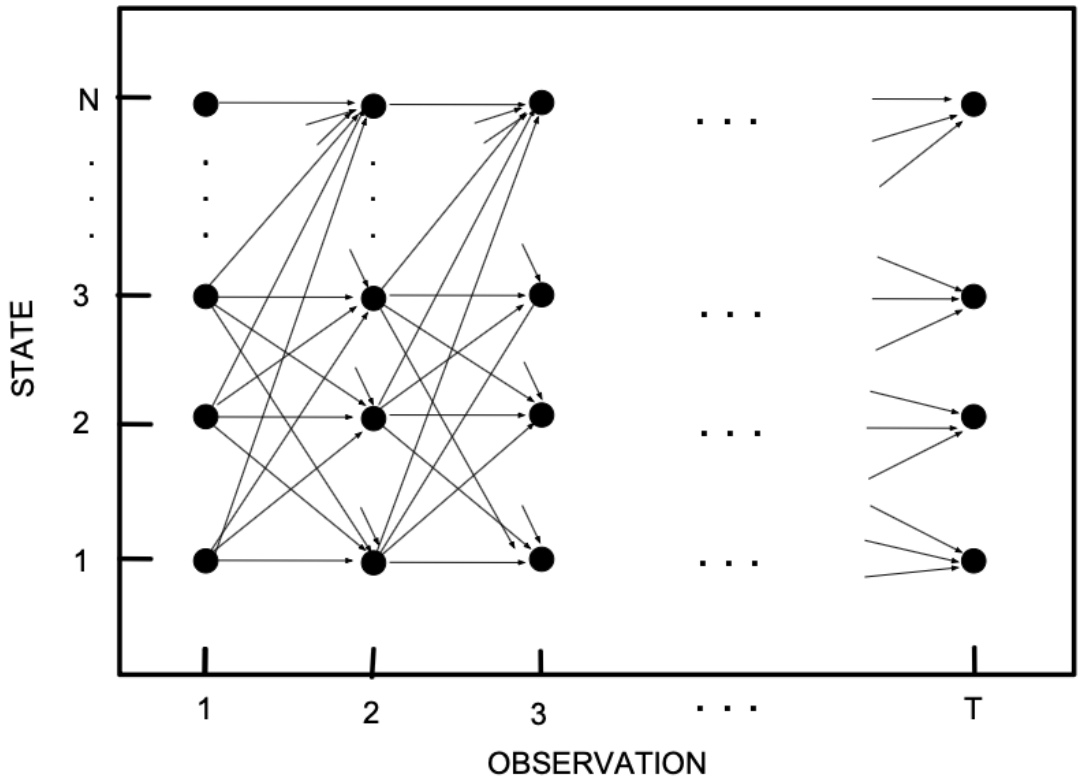
The Viterbi algorithm

Now we consider the MAP estimate, finding the sequences with maximum posterior probability:

$$\begin{aligned}\boldsymbol{z}_{1:T}^{*} &= \underset{\boldsymbol{z}_{1:T}}{\operatorname{argmax}}\, p\left(\boldsymbol{z}_{1:T} \mid \boldsymbol{y}_{1:T}\right) = \underset{\boldsymbol{z}_{1:T}}{\operatorname{argmax}} \log p\left(\boldsymbol{z}_{1:T} \mid \boldsymbol{y}_{1:T}\right) \\ &= \underset{\boldsymbol{z}_{1:T}}{\operatorname{argmax}} \log \pi_1\left(\boldsymbol{z}_1\right) + \log \lambda_1\left(\boldsymbol{z}_1\right) + \sum_{t=2}^{T}\left[\log A\left(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t\right) + \log \lambda_t\left(\boldsymbol{z}_t\right)\right]\end{aligned}$$

This is equivalent to computing a *shortest path through the trellis diagram* below, where the nodes are possible states at each time step, and

- the nodes are $\log \lambda_t\left(\boldsymbol{z}_t\right)$, except the first layer nodes are $\log \pi_1\left(\boldsymbol{z}_1\right) + \log \lambda_1\left(\boldsymbol{z}_1\right)$,
- the edges are weighted by $\log A\left(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t\right)$.



The Viterbi algorithm is a *dynamical programming* algorithm, the algorithm is as follow:

Forward Pass: The maximum probability we can assign to the data up to z_t if we end up in state j is given by the path

$$\delta_t(j) \triangleq \max_{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_{t-1}} p\left(\boldsymbol{z}_{1:t-1}, \boldsymbol{z}_t = j, \boldsymbol{y}_{1:t}\right)$$

The most probable path to state j at time t must consist of the most probable path to some other state i at time $t-1$, so

$$\delta_t(j) = \lambda_t(j)\left[\max_i \delta_{t-1}(i) A_{i,j}\right]$$

and we initialize by setting $\delta_1(j) = \pi_j \lambda_1(j)$. We also need to keep track of the most likely previous (ancestor) state, for each possible state that we end up in:

$$a_t(j) \triangleq \operatorname*{argmax}_i \delta_{t-1}(i) A_{i,j}$$

At the end of forward pass, we will end up with, for each t and j, an index $a_t(j)$.

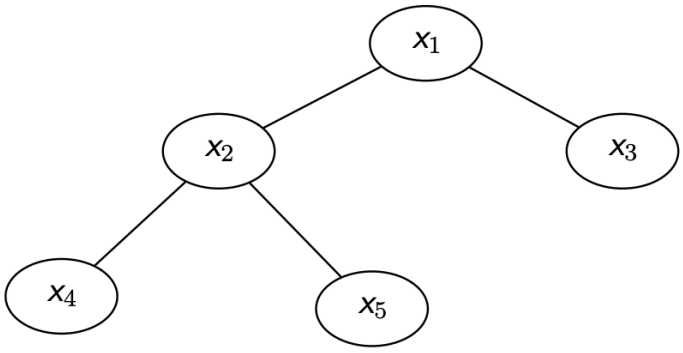
Backward pass: In the backwards pass, we *compute the most probable sequence* of states using a traceback procedure, as follows: $\boldsymbol{z}_t^* = a_{t+1}\left(\boldsymbol{z}_{t+1}^*\right)$, where we initialize using $\boldsymbol{z}_T^* = \arg\max_i \delta_T(i)$.

Overall, the time complexity of Viterbi is clearly $O\left(K^2T\right)$ in general, and the space complexity is $O(KT)$.

Belief propagation on trees

Sum-product algorithm

We assume that our model is an undirected tree, we pick an arbitrary node as a root, and orient all the edges downwards away from this root, so that each node has a unique parent:



In the graph we define the message passing:

$$
\begin{aligned}
m_{4\to 2}\left(z_2\right) &= \sum_{z_4} \psi_4\left(z_4\right)\psi_{24}\left(z_2, z_4\right) \\
m_{5\to 2}\left(z_2\right) &= \sum_{z_5} \psi_5\left(x_5\right)\psi_{25}\left(z_2, z_5\right) \\
m_{3\to 1}\left(z_1\right) &= \sum_{z_3} \psi_3\left(x_3\right)\psi_{13}\left(z_1, z_3\right) \\
m_{2\to 1}\left(z_1\right) &= \sum_{z_2} \psi_2\left(x_2\right)\psi_{12}\left(z_1, z_2\right)m_{4\to 2}\left(z_2\right)m_{5\to 2}\left(z_2\right)
\end{aligned}
$$

Finally, we can obtain the marginal distribution over z_1:

$$
p\left(z_1\right) \propto \psi_1\left(z_1\right)m_{2\to 1}\left(z_1\right)m_{3\to 1}\left(z_1\right).
$$

Upward Passing (Collect to root): *In general*, we define the **belief state** for each variable:

$$
\mathrm{bel}_s\left(z_s\right) \propto \psi_s\left(z_s\right) \prod_{t\in \mathrm{ch}_s} m_{t\to s}\left(z_s\right)
$$

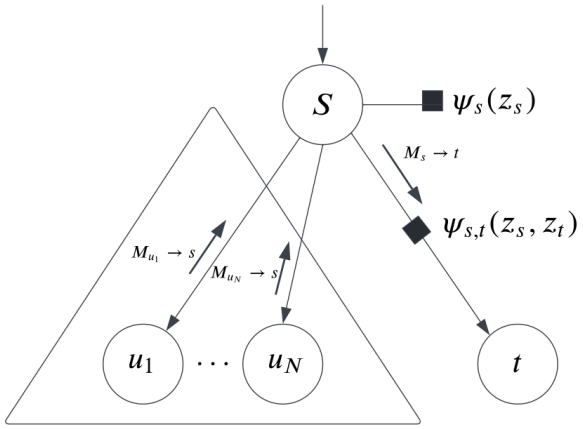
To compute the outgoing message that s should send to its parent t, we pass the local belief through the pairwise potential linking s and t, and then marginalize out s to get

$$
m_{s\to t}\left(z_t\right) = \sum_{z_s} \psi_{st}\left(z_s, z_t\right) \mathrm{bel}_s\left(z_s\right)
$$

Then, at the root of the tree, $\text{bel}_t(z_t) = p(z_t)$ will have seen all the evidence.

Downward Passing (Distribute from root): Send messages back down to the leaves. The message that s sends to its child t is given by

$$m_{s \to t}(z_t) = \sum_{z_t} \left(\psi_s(z_s) \psi_{st}(z_s, z_t) \prod_{u \in \text{ch}_s \setminus t} m_{u \to s}(z_s) \right) = \sum_{z_s} \psi_{st}(z_s, z_t) \frac{\text{bel}_s(z_s)}{m_{t \to s}(z_s)}$$



We use this to update the belief state at node t to get:

$$\text{bel}_t(z_t) \propto \text{bel}_t(z_t) m_{s \to t}(z_t)$$

Now, after both passing, we have that for each z_t:

$$\text{bel}_t(z_t) = p(z_t)$$

Max-product algorithm

We can replace the sum operation with the max operation to get max-product belief propagation. The result of this computation are a set of max marginals for each node, so we have that for each z_i:

$$\text{bel}_i^*(z_i) \propto \max_{\boldsymbol{z}_{-i}} p(z_i = k, \boldsymbol{z}_{-i} \mid \boldsymbol{y})$$

Now we can derive two different kinds of "MAP" estimates from these local quantities:

- **Maximizer of the posterior marginal (MPM):** $\hat{z}_i = \text{argmax}_{z_i} \text{bel}_i(z_i)$,
- **Maximizer of the max marginal or (MMM):** $\tilde{z}_i = \text{argmax}_{z_i} \text{bel}_i^*(z_i)$.

Let:

- $\hat{\boldsymbol{z}} = \left[\hat{z}_1, \ldots, \hat{z}_{N_z}\right]$ be the MPM estimate,
- $\tilde{\boldsymbol{z}} = [\tilde{z}_1, \ldots, \tilde{z}_{N_z}]$ be the MMM estimate,
- $\boldsymbol{z}^* = \operatorname{argmax}_{\boldsymbol{z}} p(\boldsymbol{z} \mid \boldsymbol{y})$ be the true MAP estimate.