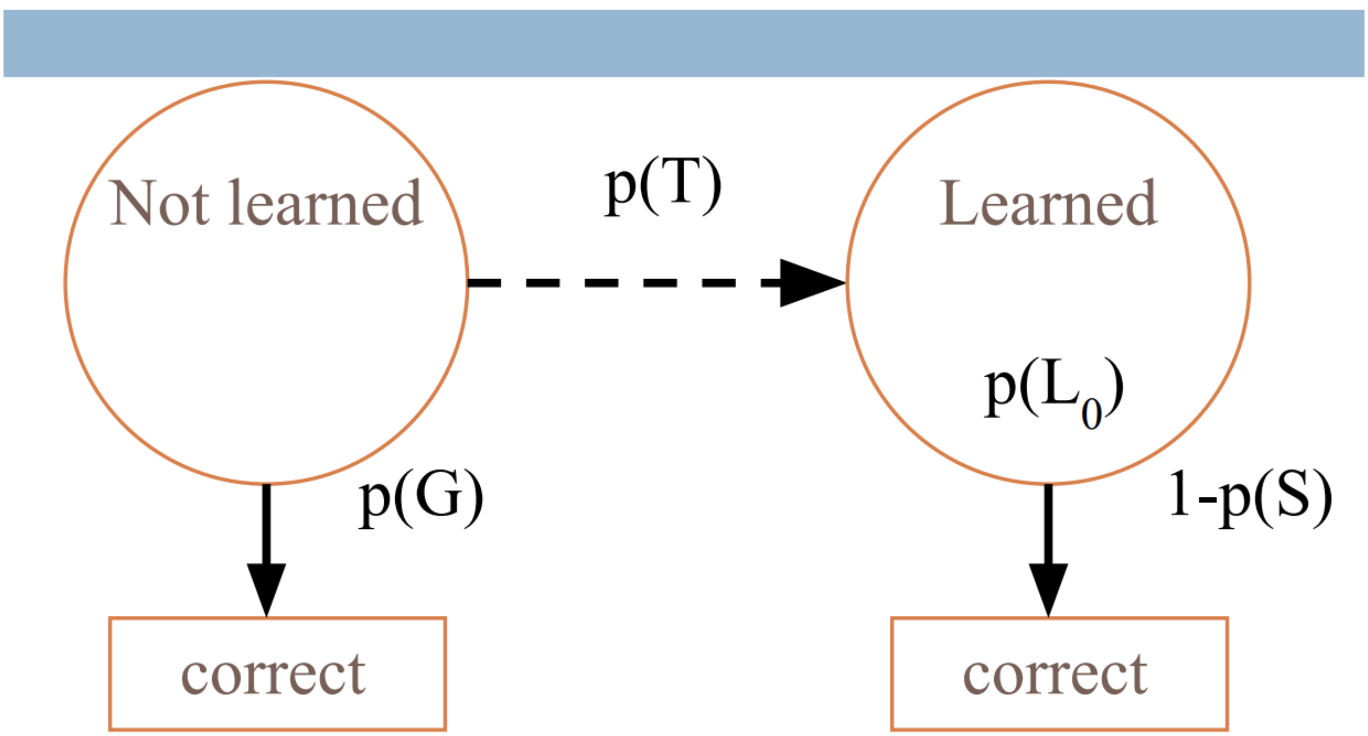
Bayesian Knowledge Tracing: Viterbi/ Baum Welch Algorithm - Organization of Python Code

A BKT performs deductions on **whether the skill(s) is (are) mastered given a sequence of correct and incorrect attempts** to solve problems (problem steps).



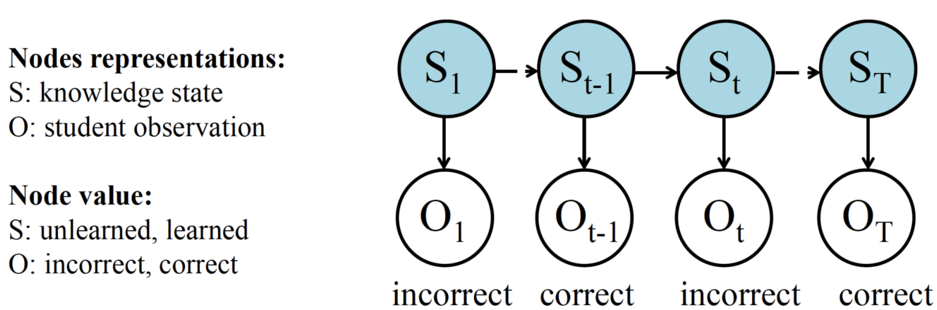


Figure 2: The Bayesian network topology of the standard Knowledge Tracing model

2 Types of nodes:

1. binary state nodes capture skill mastery (one per skill) that assume values of **mastered or not mastered**
2. binary observation nodes (one per skill) that assume values of **correct or incorrect**.

Each skill has these parameters: see

- π = $p(L_1)_u^k = p(L_0)^k$is the initial probability to master the skill
- L_t = student masters the skill at time t
- **p-transit:** T: The student's knowledge of a skill transitions from not known to known state after an opportunity to apply it
- **p-slip:** S: The probability the student makes a mistake (slip) when the skill is known
- **p-guess:** G: The student guesses correctly

And:

- **p-forget or p(F)** - is a probability that the skill will transition into *not mastered state* after a practice attempt. **Traditionally, p-forget is set to zero** and is not counted towards the total number of parameters.

Vector/Matrix-Format

Hidden States: (mastered, not mastered)
Observed States: (correct, incorrect)

Priors

$$\pi = [p(\text{knows before}) \quad 1 - p(\text{does not know before})]$$

$$\pi = [p(L_0) \quad 1 - p(L_0)]$$

Transition Probabilities A

```
               | Mastered | Not Mastered |
|--------------|----------|------------- |
| Mastered     |1         |0     (forget) |
| Not mastered |p(Transit)|1-p(Transit)  |
```

$$A = \begin{bmatrix} 1 & 0 \\ p(T) & 1 - p(T) \end{bmatrix}$$

Emission Probabilities B

```
               | Correct  | Incorrect |
|--------------|----------|-----------|
| Mastered     |1-p(slip)|p(Slip)     |
| Not mastered |p(Guess)  |1-p(Guess) |
```

$$B = \begin{bmatrix} 1 - p(S) & p(S) \\ p(G) & 1 - p(G) \end{bmatrix}$$

Dataformat

- 2009-2010 ASSISTment Data
- Source: https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data

Needed Variables:

- user_id
  - The ID of the student doing the problem.
- problem_id: The ID of the problem.
- correct
  - 1 = Correct on the first attempt

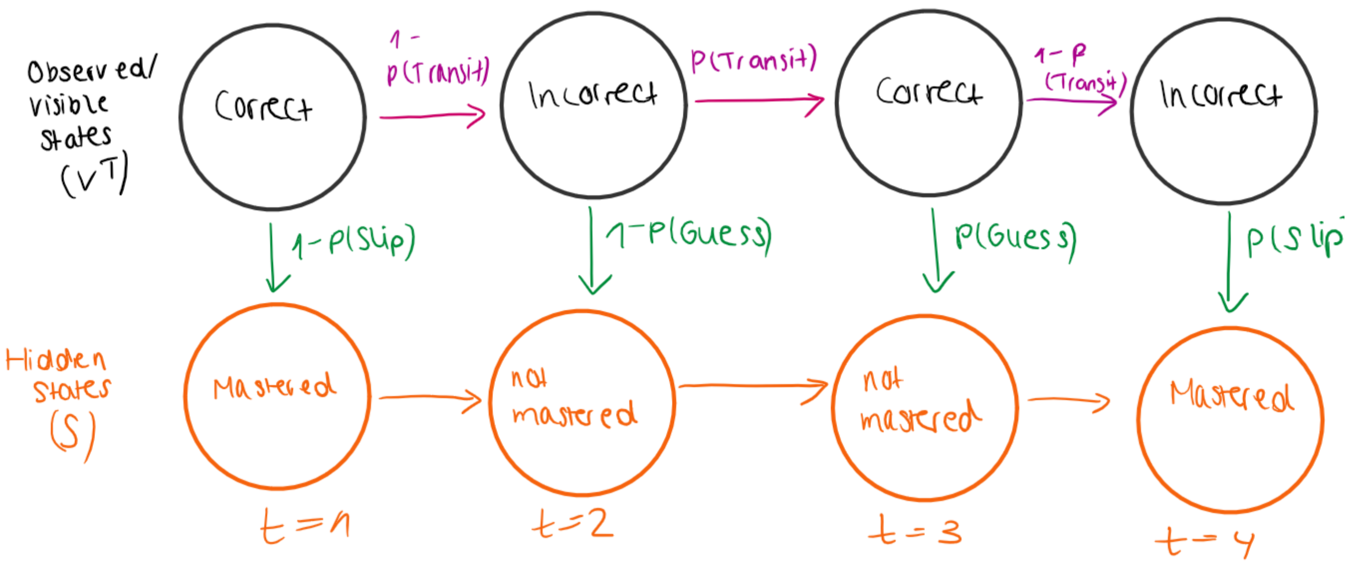
- 0 = Incorrect on the first attempt, or asked for help

→: Observation Sequence:

$Y = (Y_1 = y_1, Y_2 = y_2, \ldots, Y_T = y_t) = correct$

- skill_id
  - ID of the skill associated with the problem
- skill_name
  - Skill name associated with the problem.

Question: How do we get variable of time? Like t = 1, 2... T?



Forward Algorithm

Given a sequence of Visible state V^T, what will be the probability that the Hidden Markov Model will be in a particular hidden state s at a particular time step t?

$$\alpha_j(t) = p(v(1)\ldots v(t), s(t) = j)$$

When t = 1:

$$
\begin{aligned}
\alpha_j(1) &= p(v_k(1), s(1)=j) \\
&= p(v_k(1)|s(1)=j)p(s(1)=j) \\
&= \pi_j p(v_k(1)|s(1)=j) \\
&= \pi_j b_{jk}
\end{aligned}
$$

$$
\begin{aligned}
\text{where } \pi &= \text{ initial distribution,} \\
b_{jkv(1)} &= \text{ Emission Probability at } t=1
\end{aligned}
$$

When t = 2:

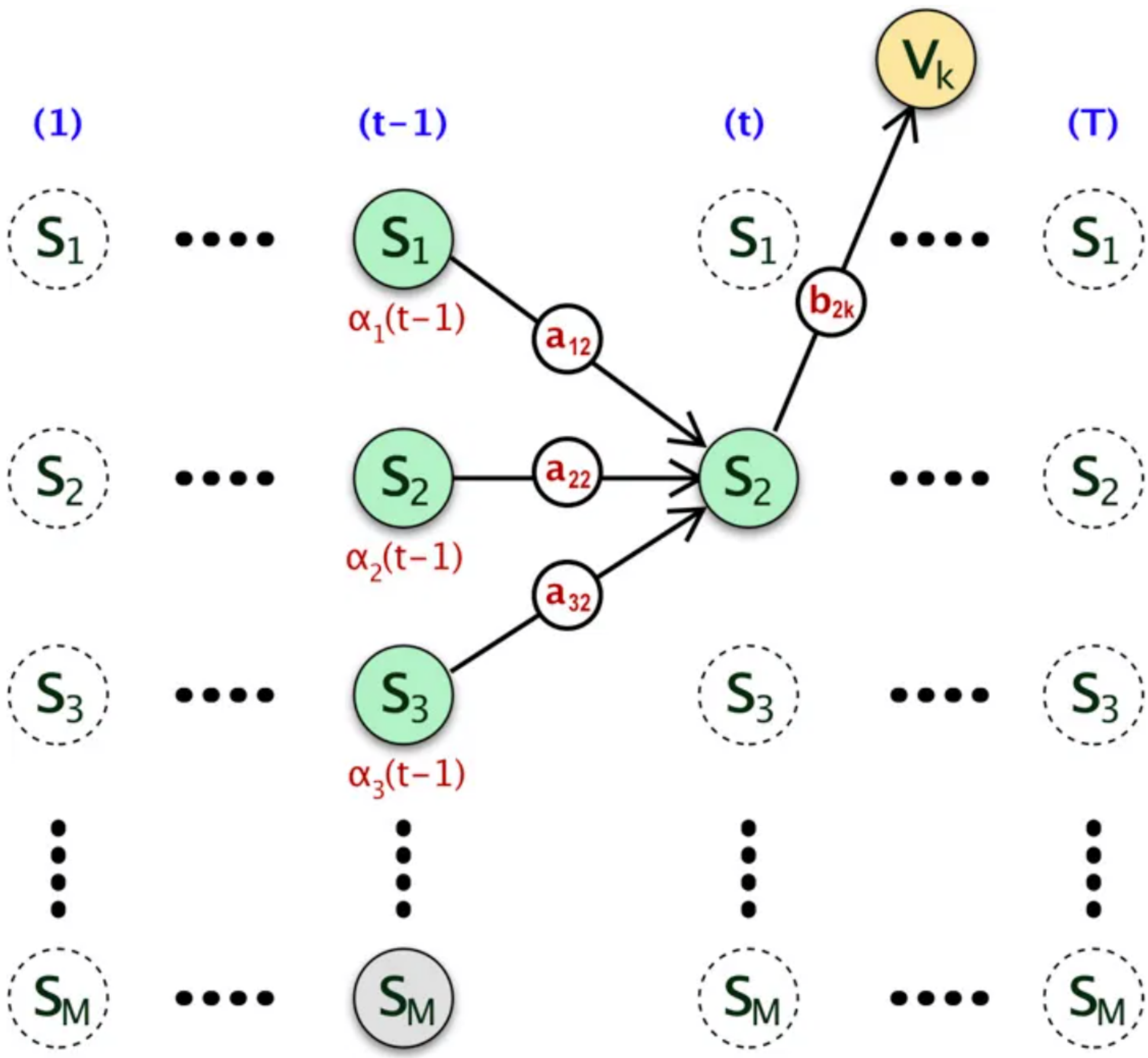
We try to implement $\alpha_j(1)$ to use recursion. There are M different Hidden States

$$
\begin{aligned}
\alpha_j(2) &= p\Big(v_k(1), v_k(2), s(2)=j\Big) \\
&= \sum_{i=1}^{M} p\Big(v_k(1), v_k(2), s(1)=i, s(2)=j\Big) \\
&= \sum_{i=1}^{M} p\Big(v_k(2)|s(2)=j, v_k(1), s(1)=i\Big) p\Big(v_k(1), s(2), s(1)=i\Big) \\
&= \sum_{i=1}^{M} p\Big(v_k(2)|s(2)=j, v_k(1), s(1)=i\Big) p\Big(s(2)|v_k(1), s(1)=i\Big) p\Big(v_k(1), s(1) \\
&= \sum_{i=1}^{M} p\Big(v_k(2)|s(2)=j\Big) p\Big(s(2)|s(1)=i\Big) p\Big(v_k(1), s(1)=i\Big) \\
&= p\Big(v_k(2)|s(2)=j\Big) \sum_{i=1}^{M} p\Big(s(2)|s(1)=i\Big) p\Big(v_k(1), s(1)=i\Big) \\
&= b_{jkv(2)} \sum_{i=1}^{M} a_{i2} \alpha_i(1)
\end{aligned}
$$

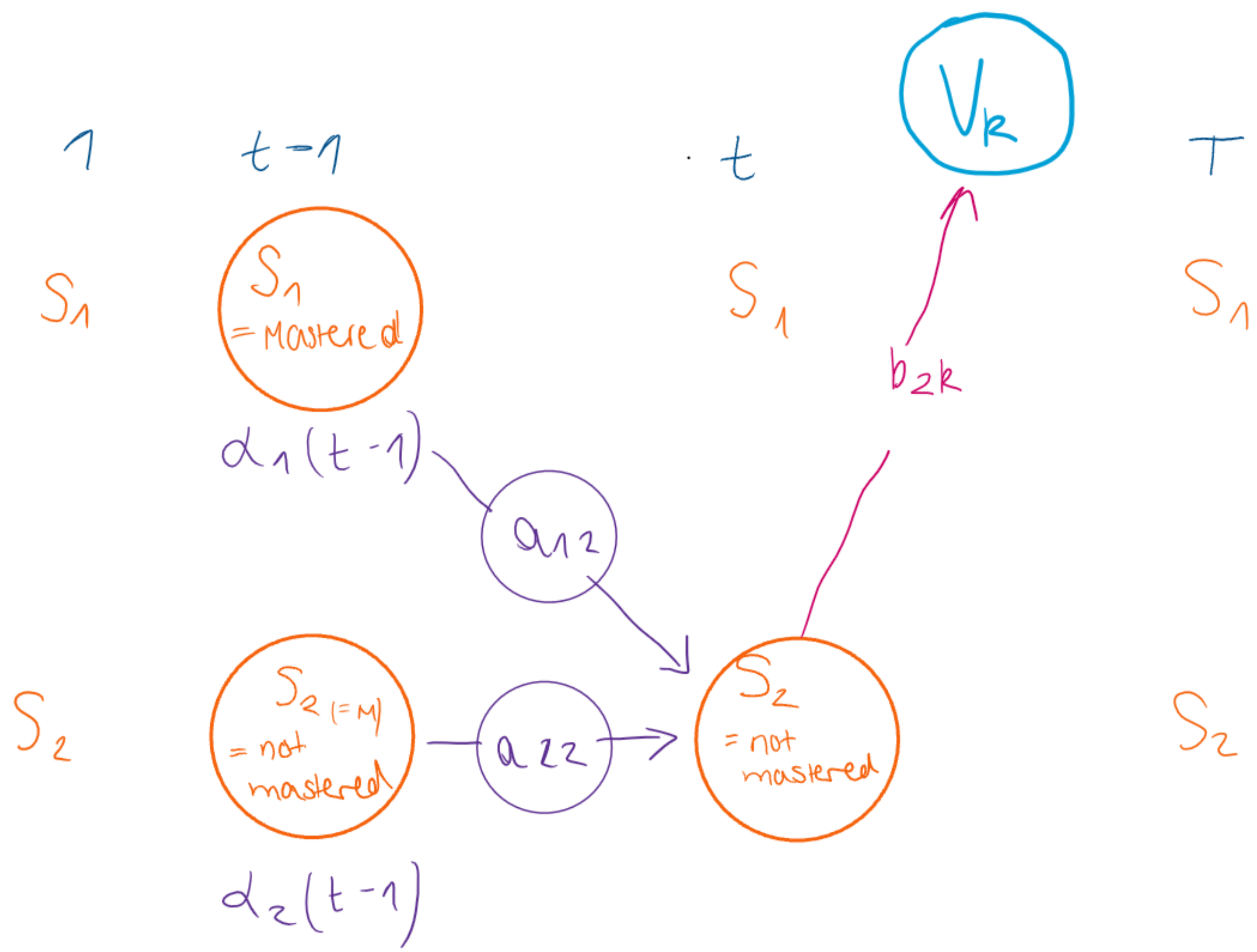
The generalized equation for any time step t+1:

$$= b_{jkv(t+1)} \sum_{i=1}^{M} a_{ij} \alpha_i(t)$$

$$\alpha_1(t-1)a_{12}$$



For our example:



$$\alpha_j(t) = \begin{cases} \pi_j b_{jk} & \text{when } t = 1 \\ b_{jk} \sum_{i=1}^{M} \alpha_i(t-1) a_{ij} & \text{when } t \text{ greater than } 1 \end{cases}$$

$\alpha_j(t)$ means the probability that the student will be at hidden state s_j (for instance mastered skill) at time step t, after emitting first t visible sequence of symbols.

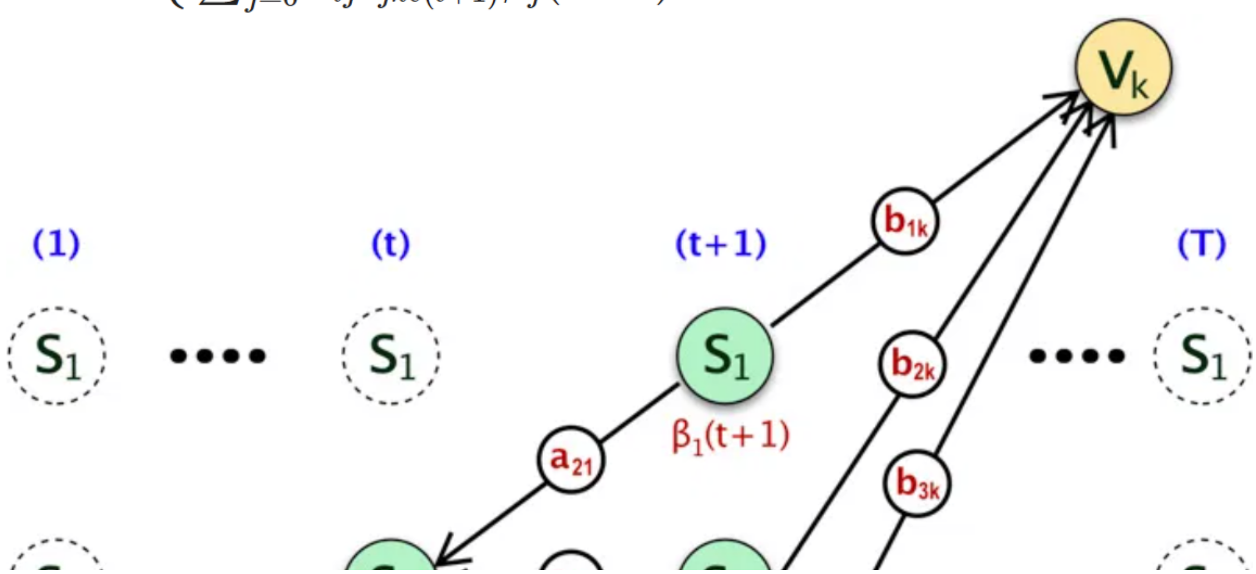
Backward Algorithm

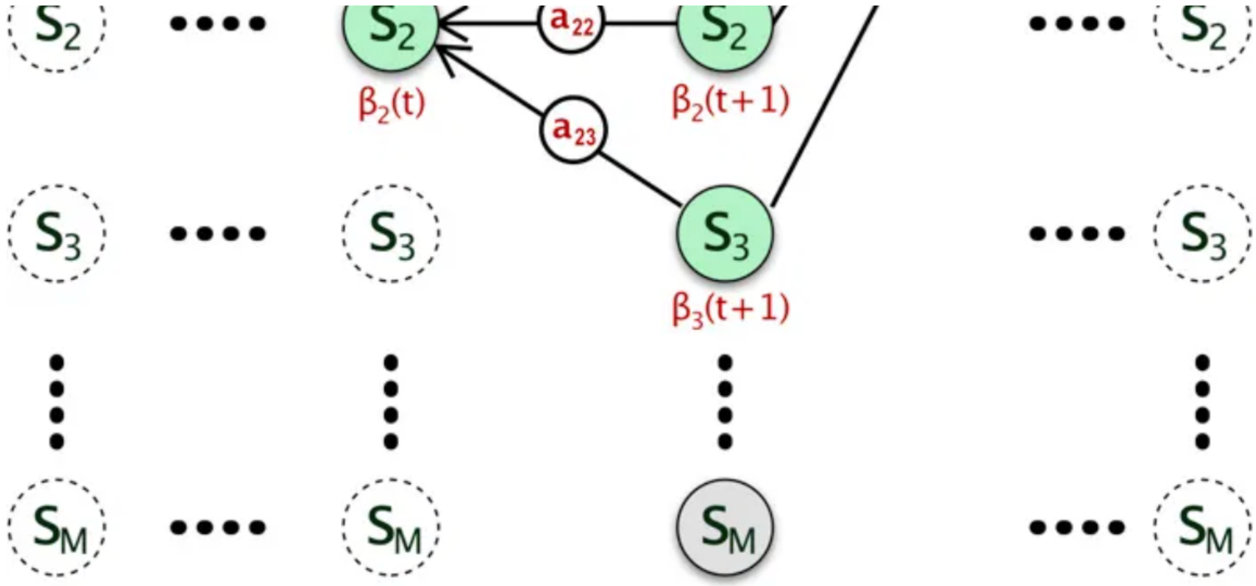
The backward algorithm is the time-reversed version of the Forward Algorithm. We need to find the probability that the student will be in hidden state s_i (for example: Mastered a skill) at time step t and will generate the remaining part of the sequence of the visible symbol V_t

$$
\begin{aligned}
\beta_i(t) &= p\Big(v_k(t+1)\ldots.v_k(T)|s(t)=i\Big) \\
&= \sum_{j=0}^{M} p\Big(v_k(t+1)\ldots.v_k(T), s(t+1)=j|s(t)=i\Big) \\
&= \sum_{j=0}^{M} p\Big(v_k(t+2)\ldots.v_k(T)|v_k(t+1), s(t+1)=j, s(t)=i\Big) \\
&\quad p\Big(v_k(t+1), s(t+1)=j|s(t)=i\Big) \\
&= \sum_{j=0}^{M} p\Big(v_k(t+2)\ldots.v_k(T)|v_k(t+1), s(t+1)=j, s(t)=i\Big) \\
&\quad p\Big(v_k(t+1)|s(t+1)=j, s(t)=i\Big)p\Big(s(t+1)=j|s(t)=i\Big) \\
&= \sum_{j=0}^{M} p\Big(v_k(t+2)\ldots.v_k(T)|s(t+1)=j\Big)p\Big(v_k(t+1)|s(t+1)=j\Big) \\
&\quad p\Big(s(t+1)=j|s(t)=i\Big) \\
&= \sum_{j=0}^{M} \beta_j(t+1)b_{jkv(t+1)}a_{ij}
\end{aligned}
$$

$$
\begin{aligned}
\text{where } a_{i2} &= \text{Transition Probability} \\
b_{jkv(t+1)} &= \text{Emission Probability at } t=t+1 \\
\beta_i(t+1) &= \text{Backward probability at } t=t+1
\end{aligned}
$$

$$
\beta_i(t) = \begin{cases} 1 & \text{when } t=T \\ \sum_{j=0}^{M} a_{ij}b_{jkv(t+1)}\beta_j(t+1) & \text{when } t \text{ less than } T \end{cases}
$$





Baum-Welch Algorithm

High-Level Steps for Baum-Welch Algorithm

1. Start with initial probability estimates `[A,B]`. A = Transition Probability Matrix, B = Emission Probability Matrix. Initially set equal probabilities or define them randomly.
2. Compute expectation of how often each transition/emission has been used. We will estimate latent variables $\sum_{t=1}^{T-1} \xi_t(i,j)$ and $\sum_{t=1}^{T-1} \gamma_t(i)$ (This is common approach for EM Algorithm)
3. Re-estimate the probabilities `[A,B]` based on those estimates (latent variable).
4. Repeat until convergence

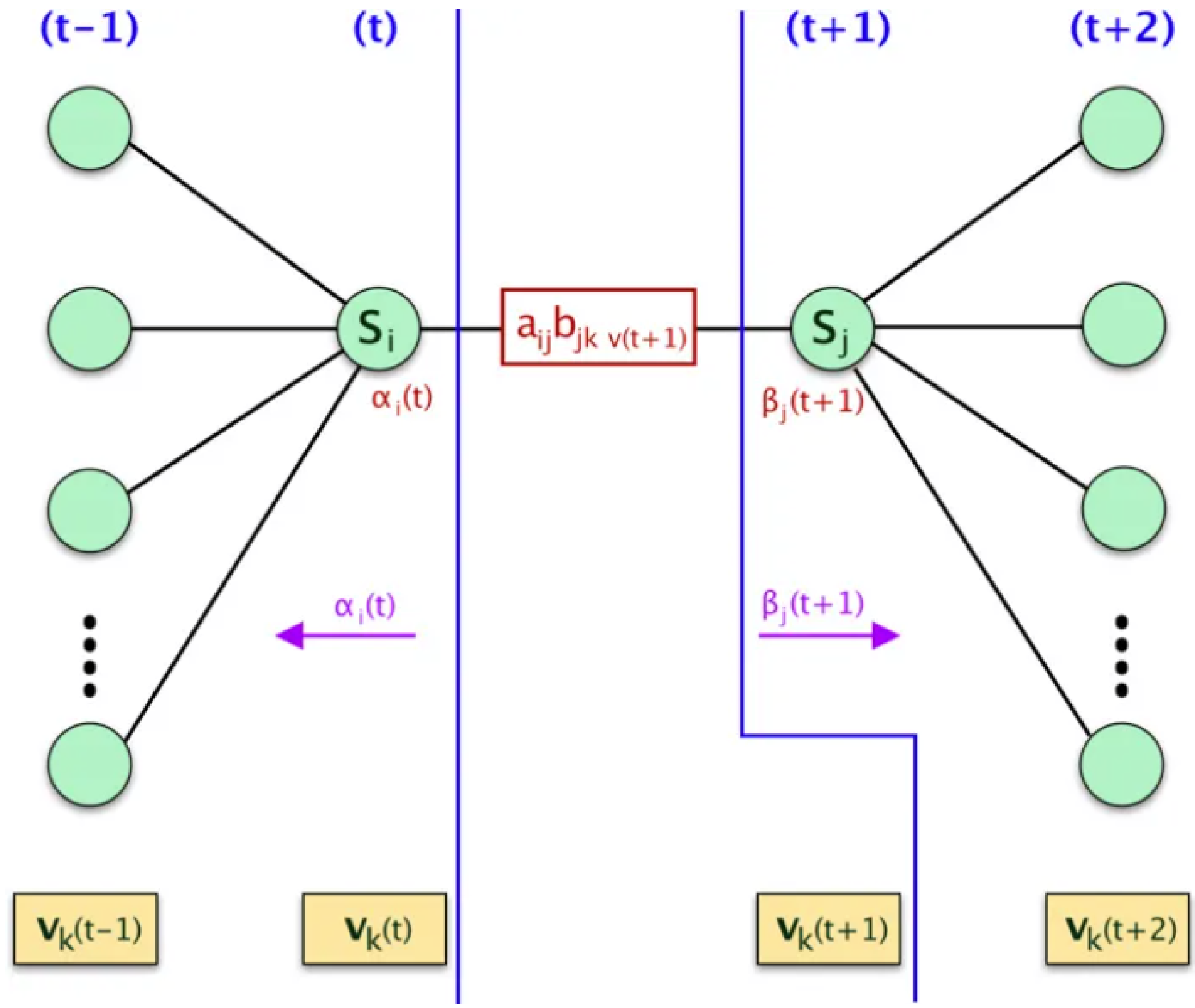
Estimate for $\hat{a_{ij}}$
$\hat{a_{ij}} = \frac{\text{expected number of transitions from hidden state i to state j}}{\text{expected number of transition from hidden state i}}$

Estimate for $\hat{b_{jk}}$
$\hat{b_{jk}} = \frac{\text{expected number of times in hidden state j and observing v(k)}}{\text{expected number of times in hidden state j}}$
For our example:

1. Correct & Mastered = 1-p(slip)

$$\frac{\text{expected number of times in hidden state "Mastered" and observing "Correct"}}{\text{expected number of times in hidden state "Mastered"}}$$

2. Correct & Not Mastered = p(slip)
3. Incorrect & Mastered = p(guess)
4. Incorrect & Not Mastered = 1-p(guess)



Final EM Algorithm:

- **initialize** A and B
- **iterate** until convergence
  - **E-Step**
    - $\xi_{ij}(t) = \frac{\alpha_i(t) a_{ij} b_{jk\, v(t+1)} \beta_j(t+1)}{\sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i(t) a_{ij} b_{jk\, v(t+1)} \beta_j(t+1}$
    - $\gamma_i(t) = \sum_{j=1}^{M} \xi_{ij}(t)$
  - **M-Step**
    - $\hat{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \sum_{j=1}^{M} \xi_{ij}(t)}$
    - $\hat{b_{jk}} = \frac{\sum_{t=1}^{T} \gamma_j(t) 1(v(t)=k)}{\sum_{t=1}^{T} \gamma_j(t)}$
- **return** A,B

Zotero links: Local library, Cloud library
http://www.adeveloperdiary.com/data-science/machine-learning/forward-and-backward-algorithm-in-hidden-markov-model/
file:///C:/Users/norap/Downloads/318-Article%20Text-1657-1-10-20181025.pdf
Tags: #HMM #BaumWelchAlgorithm #ASSISTmentData