Comparing Non-Bayesian and Bayesian approaches for Hidden Markov Models

Kevin Yang, Xiaoxuan Liang

1 Abstract

A Markov chain is a model that tells us something about the probabilities of sequences of random variables and states, each of which can take on values from some set (Jurafsky and Martin, 2020). A Hidden Markov Model, or HMM, is a generalization of a Markov chain and allows for analysis on both observed and hidden states (Fonzo et al, 2007). Currently, HMMs have become standard in statistics and are widely used in bioinformatics, econometrics and statistical signal processing (Ryden, 2008). In a 1970 paper, Baum et al. introduced an EM algorithm - commonly known as the 'Baum-Welch algorithm', to learn the model parameters of an HMM given an observation set. In recent years, we have seen an interest to answer these learning problems using a Bayesian approach. Some examples include: Bayesian Structural Inference, which relies on a set of candidate unifiliar HMM topologies for inference of process structure from a data series (Strelioff and Crutchfield, 2013); and Bayesian Baum-Welch, a model primarily applied in Bioinformatics, that integrates chromosomal spatial information to perform inference (Seifert et al., 2011). Motivated by these examples and their exceptional performance, we propose to implement a Gibbs sampler introduced by Tobias Ryden to tackle the learning problem (Ryden, 2008). Subsequently, we will contrast the Baum-Welch algorithm with Ryden's Gibbs sampler and discuss each one's limitations and strengths.

2 Background

2.1 HMM

The Hidden Markov Model (HMM) is a statistical model that captures the dependence among a sequence of n observable random variables $\mathbf{X} = \{X_1, X_2, \cdots, X_n\}$ and a corresponding sequence of n unobservable, or hidden, random variables $\mathbf{Z} = \{Z_1, Z_2, \cdots, Z_n\}$. The hidden random variables $Z_i \in \{1, 2, \cdots, K\}$ are discrete, where K is the number of hidden states. The observable random variables X_i can be either discrete or continuous. In our project, we explored the continuous case where observations originated from a Gaussian distribution. The model assumptions of an HMM are as follows,

- $\mathrm{P}\,(\,Z_k = z_k \mid Z_{k-1} = z_{k-1}, \cdots, Z_1 = z_1\,) = \mathrm{P}\,(\,Z_k = z_k \mid Z_{k-1} = z_{k-1}\,)$ (Markov property)
- $\mathrm{P}\,(\,X_k \mid X_1, \cdots, X_n, Z_1, \cdots, Z_n\,) = \mathrm{P}\,(\,X_k \mid Z_k\,)$ (observation value only depends on current state)

The structural (homogeneous) HMM contains three components,

- The initial state distribution $\pi_i = \mathrm{P}\,(\,Z_1 = z_i\,)$: the probability of beginning at state i at time point 1.
- The transition matrix $T_{ij} = \mathrm{P}\,(\,Z_t = j \mid Z_{t-1} = i\,)$: a $K \times K$ matrix that captures the probabilities of moving from state j to state i at time t.
- The emission distribution $f_{z_t}(X_t) = \mathrm{P}\,(\,X_t \mid Z_t = z_t\,)$: the conditional probability of the observable variables given being in state z_t at time t. Here, the emission distribution of continuous observable variables is Gaussian with parameters μ_k and σ_k^2.

The likelihood function for an HMM with observed data $\{x_1, x_2, \cdots x_n\}$ is,

$$\begin{aligned} \mathrm{P}\,(\,x_1, \cdots, x_n \mid \theta\,) &= \sum_{z_1, \cdots, z_n} \mathrm{P}\,(\,x_1, \cdots, x_n, z_1, \cdots, z_n \mid \theta\,) \\ &= \sum_{z_1, \cdots z_n} \mathrm{P}\,(\,z_1\,)\,\mathrm{P}\,(\,x_1 \mid z_1, \theta\,) \prod_{j=2}^{n} \mathrm{P}\,(\,z_j \mid z_{j-1}\,) \prod_{j=1}^{n} \mathrm{P}\,(\,x_j \mid z_j, \theta\,)\,. \end{aligned}$$

where $\theta = (\mu, \sigma^2)$. This sum contains K^n terms, so it becomes computationally expensive to compute when the observation set is very large. This project focused on two efficient methods, one Non-Bayesian and one Bayesian for maximizing this likelihood function.

2.2 Bayesian Inference

2.2.1 Monte Carlo

Monte Carlo aims to draw an i.i.d. set of samples $\{x_1, x_2, \ldots, x_n\}$ from a target distribution $p(x)$ defined on a high dimensional space. These samples can be used to approximate the target distribution using the following formula,

$$p(x) = \frac{1}{n} \sum_{i=1}^{N} \delta_{x^{(i)}}(x)$$

where $\delta_{x^{(i)}}(x)$ is the delta-Dirac mass located at $x^{(i)}$. Estimates from Monte Carlo methods are also unbiased and by the strong law of large numbers, almost surely (a.s.) converges.

2.2.2 Markov Chains

A Markov Chain is a stochastic model which describes a sequence of events where the probability of a specific event occurring only depends on the previous event. Alternatively, let $x^{(i)} \in \mathcal{X} = \{x_1, x_2, \ldots, x_s\}$. The stochastic process $x^{(i)}$ is a Markov chain if,

$$p(x^{(i)} \mid x^{(i-1)}, \ldots, x^{(1)}) = T(x^{(i)} \mid x^{(i-1)}), \quad \texttt{and}$$

$$\sum_{x^{(i)}} T(x^{(i)} \mid x^{(i-1)}) = 1, \quad \forall i$$

where $T(x^{(i)} \mid x^{(i-1)})$ is the transition probability.

Two important properties of a Markov Chain include,

- Irreducibility: For any state of the Markov chain, there is a positive probability of visiting all other states
- Aperiodicity: The chain does not get trapped in cycles

2.2.3 Markov Chain Monte Carlo (MCMC)

MCMC methods are a class of algorithms which use principles from Monte Carlo methods and Markov Chains to sample from an intractable posterior distribution (Ravenzwaaij et al., 2016). Unlike Monte Carlo sampling which draws independent samples from a distribution, MCMC draws samples from a Markov Chain. In MCMC, the Markov Chain is constructed such that each successive sample that is drawn from the probability distribution depends only upon the previous sample. It can be shown that an irreducible, aperiodic Markov Chain will converge to its stationary distribution no matter what initial distribution is used (Andrieu et al., 2003). This result plays a fundamental role in all MCMC sampling methods, including Gibbs sampling, which is briefly discussed in the following subsection.

2.2.4 Gibbs Sampling

Gibbs sampling is a powerful MCMC sampling procedure which draws samples for each parameter directly from its conditional distribution (Casella and George, 1992). It is common to conduct a 'burn-in' period and ignore a set of samples at the beginning of the chain. See the Appendix for more details on Gibbs Sampling.

3 Methods

3.1 Numeric Stability

A prevalent issue when implementing Hidden Markov Models is that probabilities degenerate quickly during the Forward-backward and Viterbi algorithms. Extremely small probability values lead to underflow errors. Therefore, our implementation used a method to transform probabilities to log probabilities for HMM algorithms (Mann, 2006). Details can be found in the Appendix.

3.2 Numba

MCMC algorithms are known to exhibit slow convergence (Robert et al., 2018). We used the Python package Numba (Lam et al., 2020) to accelerate our implementation of the Gibbs sampler. Numba provided a speed up of approximately two orders of magnitude.

3.3 Baum-Welch

The Baum-Welch algorithm is a popular method used to infer the parameters of a HMM. It is a special case of the EM algorithm and naturally suffers from similar limitations - one being sensitivity to initialization (Liu et al., 2021). At a high level, the Baum-Welch algorithm iterates over two steps: an E-step and an M-step (Salakhutdinov, 2011). In the E-step, the forward-backward algorithm computes the posterior distribution over the state path. In the M-step, the expected complete data log-likelihood is maximized. In this project, Baum-Welch represented the non-Bayesian approach. See the Appendix for more details.

3.4 Gibbs Sampler for HMMs

Here, we present the Gibbs sampler proposed by Tobias Ryden for an HMM (Ryden, 2008). This Gibbs sampler represented the Bayesian approach in our project. Let $(\rho_1, \rho_2, \ldots, \rho_d)$ be the initial distribution, μ_i be the mean of the emission distribution for state i, σ^{-2} be the inverse variance of the emission distribution for state i, $(a_{i1}, a_{i2}, \ldots, a_{id})$ be row i of the state transition matrix, Z_k be the hidden state for observation x_k and ϕ be the density of a Normal distribution. The Gibbs sampler is as follows,

Algorithm 1: MCMC algorithm for HMM

$\xi = \frac{\min x_k + \max x_k}{2}$, $R = \max x_k - \min x_k$, $\kappa = \frac{1}{R^2}$, $g = 0.2$, $\alpha = 2$, $h = \frac{10}{R^2}$

$(\rho_1, \rho_2, \ldots, \rho_d) \sim Dir(1, 1, \ldots, 1)$

$\mu_i \sim N(\xi, \kappa^{-1})$

$\beta \sim \Gamma(g, h)$

$\sigma^{-2} \sim \Gamma(\alpha, \beta)$

for *i=1...t* **do**

$S_i = \sum_{k:Z_k=i} x_k$, $n_i = \#\{1 \le k \le n : Z_k = i\}$, $n_{ij} = \#\{1 < k \le n : Z_{k-1} = i, Z_k = j\}$

$\mu_i \mid \ldots \sim N(\frac{S_i + \kappa\xi\sigma^2}{n_i + \kappa\sigma^2}, \frac{\sigma^2}{n_i + \kappa\sigma^2})$

$\sigma^{-2} \mid \ldots \sim \Gamma(\alpha + \frac{1}{2}n, \beta + \frac{1}{2}\sum_{k=1}^{n}(x_k = \mu_{Z_k})^2)$

$\beta \mid \ldots \sim \Gamma(g + \alpha, h + \sigma^{-2})$

$(a_{i1}, a_{i2}, \ldots, a_{id}) \mid \ldots \sim Dir(n_{i1} + 1, n_{i2} + 1, \ldots, n_{id} + 1)$

$(\rho_1, \rho_2, \ldots, \rho_d) \mid \ldots \sim Dir(I\{Z_1 = 1\} + 1, I\{Z_1 = 2\} + 1, \ldots, I\{Z_1 = d\} + 1)$

$\mathrm{P}\left(Z_1 = j \mid \ldots\right) \propto \rho_j \phi(x_1; \mu_j, \sigma^2) p_\theta(x_{2:n} \mid Z_1 = j)$

for *k=2, ..., n* **do**

$\mathrm{P}\left(Z_k = j \mid Z_{k-1} = i\right) \propto a_{ij} \phi(x_k; \mu_j, \sigma^2) p_\theta(x_{k+1:n} \mid Z_k = j)$

3.5 Ancestral Sampling

Let π be the stationary distribution of an arbitrary transition matrix. Generate a list $P_k = \sum_{i \leq k} \pi_i$. Then, sample a_1 from a standard uniform distribution. Let $z_1 = n_1$, such that P_{n_1} is the smallest entry in P that is larger than a_1. For subsequent states, construct a matrix $Q_{(i,j)} = \sum_{k \leq j} A_{(i,k)}$. For each successive state z_i, draw a_i from a standard uniform distribution and set $z_i = n_i$, such that $Q_{(z_{i-1}, n_i)}$ is the smallest entry that is larger than a_i of the z_{i-1}^{th} row of Q. Repeat until the desired number of observations is obtained.

4 Experiments on Synthetic data

4.1 Data Simulation

We simulated a sequence of 2000 observations from 6 hidden states. The first 1000 observations made up the training set, and the remaining 1000 made up the test set. Gaussian emission probabilities were used for each hidden state. The emission distribution for hidden state i was $N(\mu_i, 2^2)$, where $\mu = (0, 5, 10, 15, 20, 25)$ and $\sigma = 2$. The transition matrix and stationary distribution were,

$$A = \begin{pmatrix} 0.8 & 0.04 & 0.05 & 0.04 & 0.03 & 0.04 \\ 0.03 & 0.85 & 0.03 & 0.04 & 0.02 & 0.03 \\ 0.02 & 0.02 & 0.9 & 0.02 & 0.02 & 0.02 \\ 0.04 & 0.03 & 0.09 & 0.75 & 0.04 & 0.05 \\ 0.0 & 0.04 & 0.03 & 0.02 & 0.87 & 0.02 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.95 \end{pmatrix}, \qquad \pi = \begin{pmatrix} 0.0872 \\ 0.126 \\ 0.224 \\ 0.076 \\ 0.125 \\ 0.361 \end{pmatrix}$$

The transition matrix was designed such that adjacent observations were more likely to stay at the same state instead of transitioning to a new state. This mimicked a Bioinformatics application of inferring CNVs from read counts using HMMs (Zahn et al., 2017). The stationary distribution π is the eigenvector with eigenvalue 1 of the transition matrix. The hidden state sequence $Z = \{z_1, z_2, \cdots z_n\}$ was initialized with ancestral sampling (Schmidt, 2021). The observation sequence $X = \{x_1, x_2, \cdots, x_n\}$ was initialized by sampling $x_i \sim N(\mu_{z_i}, \sigma^2)$ where $\sigma = 2$. Figure 1 shows a KDE plot superimposed on a histogram of the observations.

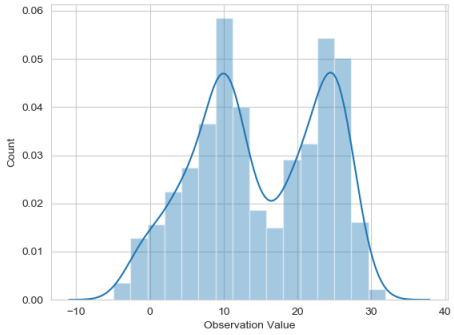


Figure 1: Simulated and estimated results on test set

4.2 Cross Validation

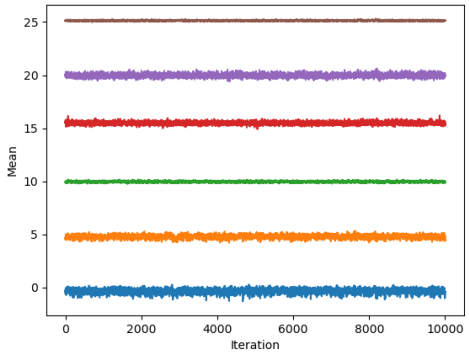
10 repetitions of 2-fold cross validation was performed for both approaches. Baum-Welch is sensitive to initialization; therefore, repetitions that had low classification rates on the test set (below 0.2) were discarded. To keep the number of repetitions consistent between approaches, we ran an extra repetition for each discarded repetition. For each repetition, the Baum-Welch algorithm terminated if the change in log likelihood was below 0.0001, and the Gibbs sampler terminated after 10100 iterations (100 burn-in). Initialization schemes for each approach are discussed in the Appendix.

4.3 Results

The mean classification rate on the training set and the test set was computed across all 10 repetitions. These results are summarized in Figure 2(a). Trace plots of each μ_i from 10000 iterations of Gibbs sampling are shown in Figure 2(b). Each distinct color in the trace plot represents a trace plot for one μ_i.

	Results
Bayesian training classification rate	0.9593
Bayesian test classification rate	0.9460
Non-Bayesian training classification rate	0.7475
Non-Bayesian test classification rate	0.7567

(a) Classification Rates



(b) 10000 iterations of Gibbs sampler

Figure 2

5 Discussion/Future Work

We first compare and contrast general limitations and strengths of each method. Some weaknesses of the Baum-Welch algorithm include: local convergence, insufficient training data, sensitivity to initialization (Rabiner, 1989), and expensive memory usage (Miklos and Meyer, 2005). There are strategies to alleviate these limitations. In practice, multiple initializations are performed to mitigate sensitivity to initialization and converging to local maxima (Biernacki et al., 2003). Addressing insufficient training data is much more difficult because obtaining more training data is usually impractical and simplifying the HMM model is typically undesirable (Rabiner, 1989). There is still work being done in this area. Perhaps the greatest strength of the Baum-Welch algorithm lies in its simplicity which provides opportunity for extension. We observe many novel algorithms that build off of Baum-Welch in the literature (Khreich et al., 2010; Miklos and Meyer, 2005; Rezaei, 2013).

One of the greatest strength for Gibbs sampling is its guarantee of convergence (Robert et al., 2018). However, significant drawbacks of Gibbs sampling include slow convergence (Robert et al., 2018) and lack of conjugacy (Schmidt, 2016). There are effective algorithmic strategies to manage these limitations. Examples of methods to speed up Gibbs sampling include: variable grouping (Bart, n.d.), auxiliary sample recycling (Martino et al., 2017), and parallel Gibbs sampling (Ko et al., 2019). Furthermore, a well-known method to address lack of conjugacy is to use the Metropolis-Hastings algorithm (Hastings, 1970; Metropolis et al., 1953).

Each method has its own strengths and weaknesses. For example, Baum-Welch is simpler to implement, while Gibbs sampling is rather complicated. In our project, the Bayesian approach was the clear winner with a mean classification rate of 0.946 on the test set. Moreover, the Gibbs sampler quickly converged to the correct mean values at the very beginning, and remained relatively consistent throughout all 10000 iterations. On the other hand, Baum-Welch achieved a mean classification rate of 0.7567 on the test set. Baum-Welch also performed poorly on some repetitions, where it attained meager classification rates below 0.2 on the test set. This clearly displayed its sensitivity to initialization. Due to time constraints, we were not able to test our model on real data. In this project, we have shown that a Bayesian approach to the learning problem can be effective. However, we recognize that there are many Non-Bayesian and Bayesian learning methods that were not explored. Therefore, this preliminary study can motivate future research to compare more complex Non-Bayesian and Bayesian approaches to the learning problem.

References

[1] Alqallaf, Fatemah, and Paul Gustafson. *On Cross-Validation of Bayesian Models.* Canadian Journal of Statistics, vol. 29, no. 2, June 2001, pp. 333–340, 10.2307/3316081. Accessed 16 Mar. 2020.

[2] Andrieu, Christophe, et al. *An Introduction to MCMC for Machine Learning. Machine Learning, vol. 50, no. 1/2, 2003, pp. 5–43, 10.1023/a:1020281327116. Accessed 29 Jan. 2020.*
`https://link.springer.com/article/10.1023%2FA%3A1020281327116`

[3] Bart, Evgeniy. *Speeding up Gibbs Sampling by Variable Grouping.*
`http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=0FD60526DE2DCDB5B259A2B8E19B6B3C?`
doi=10.1.1.682.3982&rep=rep1&type=pdf

[4] Biernacki, Christophe, et al. *Choosing Starting Values for the EM Algorithm for Getting the Highest Likelihood in Multivariate Gaussian Mixture Models.* Computational Statistics Data Analysis, vol. 41, no. 3-4, 28 Jan. 2003, pp. 561–575.
`https://www.sciencedirect.com/science/article/abs/pii/S0167947302001639.`

[5] Casella, George, and Edward I. George *Explaining the Gibbs Sampler.* The American Statistician, vol. 46, no. 3, Aug. 1992, pp. 167–174.
`www.jstor.org/stable/2685208.`

[6] Hastings, W. K. *Monte Carlo Sampling Methods Using Markov Chains and Their Applications.* Biometrika, vol. 57, no. 1, 1 Apr. 1970, pp. 97–109, 10.1093/biomet/57.1.97. Accessed 15 Oct. 2019.

[7] Khreich, Wael, et al. *On the Memory Complexity of the Forward–Backward Algorithm.* Pattern Recognition Letters, vol. 31, no. 2, 15 Jan. 2010, pp. 91–99,
`www.sciencedirect.com/science/article/abs/pii/S0167865509002578, 10.1016/j.patrec.2009.09.023.`

[8] Ko, G. G., et al. *Accelerating Bayesian Inference on Structured Graphs Using Parallel Gibbs Sampling.* IEEE Xplore, 2019.
`https://ieeexplore.ieee.org/document/8892191.`

[9] Lam, Siu Kwan, et al. *Numba.* Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM '15, 2015, 10.1145/2833157.2833162.

[10] Liu, T., et al. *Proper Initialization of Hidden Markov Models for Industrial Applications.* EEE Xplore, 2014
`https://ieeexplore.ieee.org/document/6889291.`

[11] Mann, Tobias . *Numerically Stable Hidden Markov Model Implementation.* 21 Feb. 2006.

[12] Martino, Luca, et al. *The Recycling Gibbs Sampler for Efficient Learning.* Digital Signal Processing, vol. 74, Mar. 2018, pp. 1–13, arxiv.org/pdf/1611.07056.pdf, 10.1016/j.dsp.2017.11.012.

[13] Metropolis, Nicholas, et al. *Equation of State Calculations by Fast Computing Machines.* The Journal of Chemical Physics, vol. 21, no. 6, June 1953, pp. 1087–1092, 10.1063/1.1699114.

[14] Miklós, István. *A Linear Memory Algorithm for Baum-Welch Training.* BMC Bioinformatics, vol. 6, no. 1, 2005, p. 231, 10.1186/1471-2105-6-231.

[15] Rabiner, L.R. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.* Proceedings of the IEEE, vol. 77, no. 2, 1989, pp. 257–286, doi:10.1109/5.18626.

[16] Rezaei, Vahid, et al. *Generalized Baum-Welch Algorithm Based on the Similarity between Sequences.* PLoS ONE, vol. 8, no. 12, 20 Dec. 2013, p. e80565, 10.1371/journal.pone.0080565.

[17] Robert, Christian P., et al. *Accelerating MCMC Algorithms." Wiley Interdisciplinary Reviews: Computational Statistics.* vol. 10, no. 5, 13 June 2018, p. e1435, 10.1002/wics.1435.

[18] Seifert, Michael, et al. *Exploiting Prior Knowledge and Gene Distances in the Analysis of Tumor Expression Profiles with Extended Hidden Markov Models.* Bioinformatics, vol. 27, no. 12, 21 Apr. 2011, pp. 1645–1652.
`https://academic.oup.com/bioinformatics/article/27/12/1645/256543.`

[19] Strelioff, Christopher C., and James P. Crutchfield. *Bayesian Structural Inference for Hidden Processes.* Physical Review E, vol. 89, no. 4, 10 Apr. 2014, p. 042119.
https://arxiv.org/abs/1309.1392.

[20] van Ravenzwaaij, Don, et al. *A Simple Introduction to Markov Chain Monte–Carlo Sampling.* Psychonomic Bulletin Review, vol. 25, no. 1, 11 Mar. 2016, pp. 143–154.
https://link.springer.com/article/10.3758%2Fs13423-016-1015-8.

[21] Baum, Leonard E., et al. *A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains.* The Annals of Mathematical Statistics, vol. 41, no. 1, 1970, pp. 164–171. JSTOR
https://www.jstor.org/stable/2239727.

[22] Salakhutdinov, R. (2011). STA 4273H: Statistical Machine Learning [PowerPoint slides]. Retrieved from the University of Toronto.

[23] Schmidt, M. (2016) MCMC and Non-Parametric Bayes [PowerPoint slides]. Retrieved from the University of British Columbia.

[24] Schmidt, M. (2021) Monte Carlo Methods [PowerPoint slides]. Retrieved from the University of British Columbia.

[25] Zahn, Hans et al. *Scalable whole-genome single-cell library preparation without preamplification.* Nature Methods, vol. 14, Feb. 2017, pp.167-173. https://pubmed.ncbi.nlm.nih.gov/28068316/

Appendix

Code Availability

https://github.com/keviny2/STAT-520A-Project

Gibbs Sampling

Suppose we have a vector $X = (x_1, \ldots, x_n)$ from a joint distribution $p(x_1, \ldots, x_n)$. Assume we have an initial sample $X^{(1)} = (x_1^{(1)}, \ldots, x_n^{(1)})$. The Gibbs sampling algorithm is as follows:

Algorithm 2: Gibbs Sampler

Begin with $X^{(1)} = (x_1^{(1)}, \ldots, x_n^{(1)})$

for *i=1 ... n* **do**

$x_1^{(i+1)} \sim p(x_1 \mid x_2^{(i)}, x_3^{(i)} \ldots, x_n^{(i)})$

$x_2^{(i+1)} \sim p(x_1 \mid x_3^{(i)}, x_4^{(i)} \ldots, x_n^{(i)})$

$\ldots$

$x_n^{(i+1)} \sim p(x_n \mid x_1^{(i)}, x_2^{(i)} \ldots, x_{n-1}^{(i)})$

Numerically Stable Implementation

By convention in Python, 'None' represents a null variable or object.

We evaluate whether x is equal to zero or less than zero or larger than zero before applying the logarithm:

Algorithm 3: compute ln(x)

if $x = 0$ **then**
  return *None*;
else if $x > 0$ **then**
  return $\ln(x)$;
else
  return *Negative input error*;

To convert the exponent value of x , we evaluate whether x is null or not before applying exponent on x:

Algorithm 4: compute exp(x)

if x = *None* **then**
  return *0*;
else
  return $\exp(x)$;

To compute $\ln(x+y)$ with $\ln(x)$ and $\ln(y)$, we evaluate if either $\ln(x)$ or $\ln(y)$ is null and return the appropriate value if either of them are. The Log-Sum-Exp trick is applied to avoid potential overflow or underflow problems as:

Algorithm 5: compute lnsum(ln(x), ln(y)

```
if ln(x) = None or ln(y) = None then
    if ln(x) = None then
        return ln(y);
    else
        return ln(x);
else
    if ln(x) > ln(y) then
        return ln(x) + ln(1 + exp(ln(y) − ln(x)));
    else
        return ln(y) + ln(1 + exp(ln(x) − ln(y)));
```

which can be verified by:

$$\begin{aligned}\ln(x) + \left(\ln(1+e^{\ln(y)-\ln(x)})\right) &= \ln(x) + ln\left(1+e^{\ln(y/x)}\right) = \ln(x) + \ln\left(1+\frac{y}{x}\right) \\ &= \ln(x) + \ln\left(\frac{x+y}{x}\right) = \ln(x) + \ln(x+y) - \ln(x) \\ &= \ln(x+y)\end{aligned}$$

To compute $\ln(xy)$ with $\ln(x)$ and $\ln(y)$, we evaluate whether $\ln(x)$ and $\ln(y)$ are null or not before adding these two terms together,

Algorithm 6: compute lnproduct(ln(x), ln(y))

```
if ln(x) = None or ln(y) = None then
    return None;
else
    return ln(x) + ln(y);
```

Baum-Welch Algorithm

The algorithm begins with initializing model parameters θ^t based on some prior distributions. The E-step computes $Q(\theta \mid \theta^t)$, the expectation of the complete log-likelihood $\mathrm{E}_{Z_1,\cdots,Z_n \mid X_1,\cdots,X_n,\theta^t}(\log \mathrm{P}(X_1,\cdots,X_n,Z_1,\cdots,Z_n \mid \theta)$ given the current estimates of the parameters θ^t with respect to the current conditional distribution of hidden variables $\{Z_1,\cdots,Z_n\}$ given the observed variables $\{X_1,\cdots,X_n\}$. The M-step maximizes the expectation to generate new parameters $\theta^{[t+1]}$. Baum-Welch iterates between the E-step and the M-step to update θ until the expected complete log-likelihood converges. In this problem, this likelihood is obtained by

$$\begin{aligned}Q(\theta \mid \theta^t) &= \mathrm{E}_{Z_1,\cdots,Z_n \mid X_1,\cdots,X_n,\theta^t}(\log \mathrm{P}(X_1,\cdots,X_n,Z_1,\cdots,Z_n \mid \theta) \\ &= \sum_{k=1}^{K} \gamma_1(k) \log \mathrm{P}(Z_1 = k) + \sum_{i=2}^{n}\sum_{j=1}^{K}\sum_{k=1}^{K} \xi_i(j,k) \log \lambda_{jk} + \sum_{i=1}^{M}\sum_{k=1}^{K} \gamma_i(k) \log \mathrm{P}\left(X_i \mid \mu_k, \sigma_k^2\right)\end{aligned}$$

where

$$\gamma_m(k) = \mathrm{P}\left(Z_m = k \mid X_1,\cdots X_n,\theta^t\right), \qquad \xi_m(j,k) = \mathrm{P}\left(Z_{m-1} = j, Z_m = k \mid X_1,\cdots,X_n,\theta^t\right)$$

This likelihood function clearly depends on all γs' and ξs', which can be solved efficiently by forward-backward algorithm introduced in the next section.

Forward-backward Algorithm

The forward-backward algorithm is an inference algorithm that allows to efficiently compute the posterior marginal distribution via dynamic programming with forward and backward messages, which reduces the time complexity from $O(K^n)$ to $O(K^2 n)$. The forward message M_j summarizes all relevant information about the past at time j

and uses the Markov property to write M_j recursively in terms of M_{j-1}. The backward message V_j summarizes all relevant information about the future at time j and uses the Markov property to write V_j recursively in terms of V_{j+1}. Specifically,

$$M_j(z_j) = \mathrm{P}\,(\,X_1, X_2, \cdots, X_j, Z_j = z_j\,) = \sum_{z_{j-1}} \mathrm{P}\,(\,z_j \mid z_{j-1}\,)\,\mathrm{P}\,(\,X_j \mid z_j\,)\,M_{j-1}(z_{j-1})$$

$$V_j(z_j) = \mathrm{P}\,(\,X_{j+1}, X_{j+2}, \cdots, X_n \mid Z_j = z_j\,) = \sum_{z_{j+1}} \mathrm{P}\,(\,z_{j+1}\,)\,\mathrm{P}\,(\,X_j \mid z_j\,)\,V_{j+1}(z_{j+1})$$

$M_1(z_1) = \mathrm{P}\,(\,z_1\,)\,\mathrm{P}\,(\,x_1 \mid z_1\,), V_n(z_n) = 1$, which are initial forward and backward messages.

The forward-backward algorithm claims that

$$\gamma_j(z_j) = \frac{M_j(z_j)V_j(z_j)}{\mathrm{P}\,(\,X\,)}, \qquad \xi_m(j,k) = \frac{M_{m-1}(j)\,\mathrm{P}\,(\,X_m \mid Z_m = k\,)\,\mathrm{P}\,(\,Z_m = k \mid Z_{m-1} = j)V_m(k)\,)}{\mathrm{P}\,(\,X\,)}$$

where $\mathrm{P}\,(\,X\,) = \sum_{i=1}^{k} M_j(i)V_j(i)$. The initial state distribution is updated by

$$\pi_k^{[t+1]} = \mathrm{P}\,(\,Z_1 = k\,) = \frac{\gamma_1(k)}{\sum_{i=1}^{k}\gamma_1(i)}$$

Gaussian parameters $\theta = (\mu, \sigma^2)$ are updated by,

$$\mu_k^{[t+1]} = \frac{1}{N_k}\sum_{i=1}^{n}\gamma_i(k)x_i, \qquad \sigma_k^{2\,[t+1]} = \frac{1}{N_k}\sum_{i=1}^{n}\gamma_i(k)(x_1 - \mu_k^{[t+1]})^2$$

where $N_k = \sum_{i=1}^{n}\gamma_i(k)$.

To show that $\gamma_m(z_m) = \mathrm{P}\,(\,Z_m = z_m \mid X_1, X_2, \cdots, X_n\,) \propto M_m(z_m)V_m(z_m)$, we express $M_m(z_m)$ and $V_m(z_m)$ as

$$\begin{aligned}
M_m(z_m) &= \mathrm{P}\,(\,X_1, \cdots, X_m, Z_m = z_m\,) \\
&= \mathrm{P}\,(\,X_m \mid Z_m = z_m\,)\,\mathrm{P}\,(\,X_1, \cdots, X_{m-1} \mid Z_m = z_m\,)\,\mathrm{P}\,(\,Z_m = z_m\,) \\
&= \mathrm{P}\,(\,X_m \mid Z_m = z_m\,)\,\mathrm{P}\,(\,X_1, \cdots, X_{m-1}, Z_m = z_m\,) \\
&= \mathrm{P}\,(\,X_m \mid Z_m = z_m\,)\sum_{z_{m-1}}\mathrm{P}\,(\,X_1, \cdots, X_{m-1}, Z_m = z_m, Z_{m-1} = z_{m-1}\,) \\
&= \mathrm{P}\,(\,X_m \mid Z_m = z_m\,)\sum_{z_{m-1}}\mathrm{P}\,(\,X_1, \cdots, X_{m-1}, Z_m = z_m \mid Z_{m-1} = z_{m-1}\,)\,\mathrm{P}\,(\,Z_{m-1} = z_{m-1}\,) \\
&= \mathrm{P}\,(\,X_m \mid Z_m = z_m\,)\sum_{z_{m-1}}\mathrm{P}\,(\,X_1, \cdots, X_{m-1} \mid Z_{m-1} = z_{m-1}\,)\,\mathrm{P}\,(\,Z_m = z_m \mid Z_{m-1} = z_{m-1}\,)\,\mathrm{P}\,(\,Z_{m-1} = z_{m-1}\,) \\
&= \mathrm{P}\,(\,X_m \mid Z_m = z_m\,)\sum_{z_{m-1}}\mathrm{P}\,(\,X_1, \cdots, X_{m-1}, Z_{m-1} = z_{m-1}\,)\,\mathrm{P}\,(\,Z_m = z_m \mid Z_{m-1} = z_{m-1}\,) \\
&= \mathrm{P}\,(\,X_m \mid Z_m = z_m\,)\sum_{z_{m-1}} M_{m-1}(z_{m-1})\,\mathrm{P}\,(\,Z_m = z_m \mid Z_{m-1} = z_{m-1}\,) \\
V_m(z_m) &= \mathrm{P}\,(\,X_{m+1}, \cdots, X_m \mid Z_m = z_m\,) \\
&= \sum_{z_{m+1}}\mathrm{P}\,(\,X_{m+1}, \cdots, X_n, Z_{m+1} = z_{m+1} \mid Z_m = z_m\,) \\
&= \sum_{z_{m+1}}\mathrm{P}\,(\,X_{m+1}, \cdots, X_n \mid Z_{m+1} = z_{m+1}, Z_m = z_m\,)\,\mathrm{P}\,(\,Z_{m+1} = z_{m+1} \mid Z_m = z_m\,) \\
&= \sum_{z_{m+1}}\mathrm{P}\,(\,X_{m+1}, \cdots, X_m \mid Z_{m+1} = z_{m+1}\,)\,\mathrm{P}\,(\,Z_{m+1} = z_{m+1} \mid Z_m = z_m\,) \\
&= \sum_{z_{m+1}}\mathrm{P}\,(\,X_{m+2}, \cdots, X_n \mid Z_{m+1} = z_{m+1}\,)\,\mathrm{P}\,(\,X_{m+1} \mid Z_{m+1} = z_{m+1}\,)\,\mathrm{P}\,(\,Z_{m+1} = z_{m+1} \mid Z_m = z_m\,) \\
&= \sum_{z_{m+1}} V_m(z_m)\,\mathrm{P}\,(\,X_{m+1} \mid Z_{m+1} = z_{m+1}\,)\,\mathrm{P}\,(\,Z_{m+1} = z_{m+1} \mid Z_m = z_m\,)
\end{aligned}$$

Therefore,

$$\begin{aligned}\gamma_m(z_m) &= \mathrm{P}\left(Z_m = z_m \mid X_1, X_2, \cdots, X_n\right) \propto \mathrm{P}\left(X_1, X_2, \cdots, X_n, Z_m = z_m\right) \\ &= M_m(z_m)V_m(z_m) \\ \Rightarrow \gamma_m(z_m) &\propto M_m(z_m)V_m(z_m)\end{aligned}$$

Viterbi Algorithm

The Viterbi algorithm is a dynamic programming algorithm to obtain the most likely sequence of hidden latent states given an observed sequence X and parameters θ. Define

$$\delta_t(i) = \max_{z_1, z_2, \cdots, z_{n-1}} \mathrm{P}\left(Z_1 = z_1, Z_2 = z_2, \cdots, Z_{n-1} = z_{n-1}, Z_n = z_n, X_1, X_2, \cdots, X_n \mid \theta\right)$$

$\delta_t(i)$ is the hidden state sequence which maximizes the likelihood.

Algorithm 7: Viterbi Algorithm

Initialization: $\delta_1(i) = \mathrm{P}\left(Z_1 = i\right)\mathrm{P}\left(X_1 \mid Z_1 = i\right)$ where $1 \le i \le K$;
for $t = 2, \cdots n$ **do**
  $\delta_t(j) \leftarrow \max_{1\le i\le K}(\delta_{t-1}(i)\,\mathrm{P}\left(Z_t = j \mid Z_{t-1} = i\right)\mathrm{P}\left(X_t \mid Z_t = j\right))$ *where* $1 \le j \le K$;
  $\phi_t(j) \leftarrow \arg\max_{1\le i\le K}(\delta_{t-1}(i)\,\mathrm{P}\left(Z_t = j \mid Z_{t-1} = i\right)\mathrm{P}\left(X_t \mid Z_t = j\right))$ *where* $1 \le j \le K$;
$P^* \leftarrow \max_{1\le i\le K} \delta_n(i)$;
$q_n^* \leftarrow \arg\max_{1\le i\le K} \delta_n(i)$;
for $t =$ *n-1, n-2,* $\cdots, 1$ **do**
  $q_t^* \leftarrow \phi_{t+1}(q_{t+1}^*)$
return $\{q_1^*, q_2^*, \cdots q_n^*\}$

Initialization Schemes

Initialization for Baum-Welch is as follows. The observation set is first divided into 6 equal intervals. The center and the length of each interval is set to be the initial mean and standard deviation of each hidden state's Gaussian distribution. The initial transition matrix has 0.8 on the diagonal and 0.04 elsewhere. The initial state distribution is initialized by calculating the proportion of observations falling within each of the 6 intervals.

Initialization for the Gibbs sampler can be found in Section 2 of Ryden's paper (Ryden, 2008).