Hidden Markov Model Analysis

Andy Pack

EEEM030
January 2021
Department of Electrical and Electronic Engineering
Faculty of Engineering and Physical Sciences
University of Surrey

Abstract

An analysis of a continuous probability density hidden Markov model is presented. Forward and backward likelihoods for all observation/state combinations are given before confirming the same observation likelihood from each method. The occupation likelihoods for each state at each time step are calculated and used in combination with the transition likelihoods to complete an iteration of Baum-Welch training. This procedure results in a re-estimation of the output function parameters, these are compared to the originals for analysis.

Contents

List of Figures

List of Tables

Listings

1 Introduction

The Markov model is a stochastic model for analysing random processes. The model describes a system of multiple states and the transitions between them; a key feature is the *Markov property* which states that the future state of the system depends only on the current state [1]. This is also described as the system being *memoryless*. The hidden Markov model extends the standard Markov chain by modelling a system where the states are not directly observable but are hidden [2], instead the states are inferred from the state topology and visible observations.

1.1 Brief

This work analyses a continuous-density hidden Markov model of 2 emitting states describing a 1D observation space. Various statistics and calculations will be analysed including forward, backward, occupation and transition likelihoods. Using these, an iteration of Baum-Welch training [2], [3] will be completed resulting in re-estimated output probabilities.

The model is specified as follows; the transition probabilities are described by entry (π) and exit (η) probabilities for each state along with a matrix of state-to-state transitions (a),

$$A = \left\{\pi_j, a_{ij}, \eta_i\right\}$$

$$\pi_j = \{0.44, 0.56\} \qquad a_{ij} = \begin{bmatrix} 0.92 & 0.06 \\ 0.04 & 0.93 \end{bmatrix} \qquad \eta_i = \{0.02, 0.03\}$$

These, in combination, describe a state topology than can be seen graphically in figure 1.

As previously mentioned, the states are described by 1D continuous probability density functions (PDF) of a Gaussian profile. Each has an associated mean and variance as seen below,

State, i	1	2
Mean, μ_i	1.00	4.00
Variance, Σ_i	1.44	0.49

Finally, a set of observations from the above model are provided below,

$$O = \{3.8, 4.2, 3.4, -0.4, 1.9, 3.0, 1.6, 1.9, 5.0\}$$

These are used as the basis for training the model using the Baum-Welch algorithm [2], [3], a form of expectation-maximisation.

2 Implementation

The work was completed using Python [4], Matplotlib [5] and the Jupyter notebook [6]. Apart from the standard library, NumPy [7] was used for a n-dimensional array implementation. An implementation of the Gaussian function [8] was written in order to derive output probability densities for each state given an observation, see listing 2.

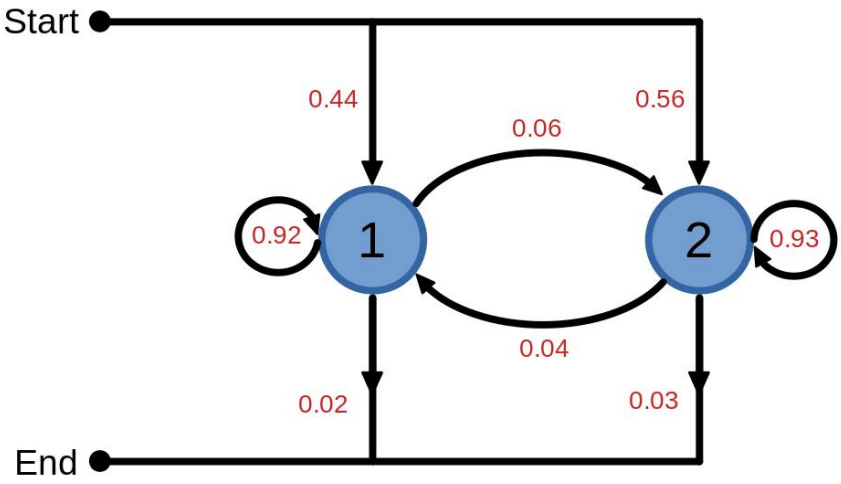

Figure 1: State topology for the specified Markov Model

Initialisation:

$$\alpha_1(i) = \pi_i b_i(o_1) \qquad 1 \leq i \leq N$$

Iteration:

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij}\right] b_j(o_t) \qquad 1 \leq j \leq N$$

Finalise:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)\, \eta_i$$

Figure 2: Forward procedure [2], [9]

Initialisation:

$$\beta_T(i) = \eta_i \qquad 1 \leq i \leq N$$

Iteration:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1})\, \beta_{t+1}(j) \qquad 1 \leq i \leq N$$

Finalise:

$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1)\, \beta_1(i)$$

Figure 3: Backward procedure [2], [9]

The forward and backward procedure are outlined by [2], [9], the algorithm for both forward and backward likelihood can be seen in figures 2 and 3. Following these procedures, the values were used to calculate the transition and occupation likelihoods [2], [3] as described in figures 4 and 5, this constitutes the E-step of the training process [10]. With all likelihoods calculated, the PDF parameters were re-estimated as described in figures 6 and 7, this is one half of the M-step of the training procedure [10].

3 Results

3.1 Output Probability Density Functions

The Gaussian probability density functions described by the initial parameters can be seen in figure 8.

$$\gamma_t(i) = P(x_t = i|O, \lambda)$$

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

Figure 4: Occupation likelihood [2], [3]

$$\xi_t(i,j) = P(x_{t-1} = i, x_t = j|O, \lambda)$$

$$\xi_t(i,j) = \frac{\alpha_{t-1}(i)a_{ij}b_j(o_t)\beta_t(j)}{P(O|\lambda)}$$

Figure 5: Transition likelihood [2], [3]

$$\hat{\mu}_i = \frac{\sum_{t=1}^{T}\gamma_t(i)o_t}{\sum_{t=1}^{T}\gamma_t(i)}$$

Figure 6: Mean re-estimation [10]

$$\hat{\Sigma}_i = \frac{\sum_{t=1}^{T}\gamma_t(i)(o_t - \mu_i)^2}{\sum_{t=1}^{T}\gamma_t(i)}$$

Figure 7: Variance re-estimation adapted for uni-variate observations [10]

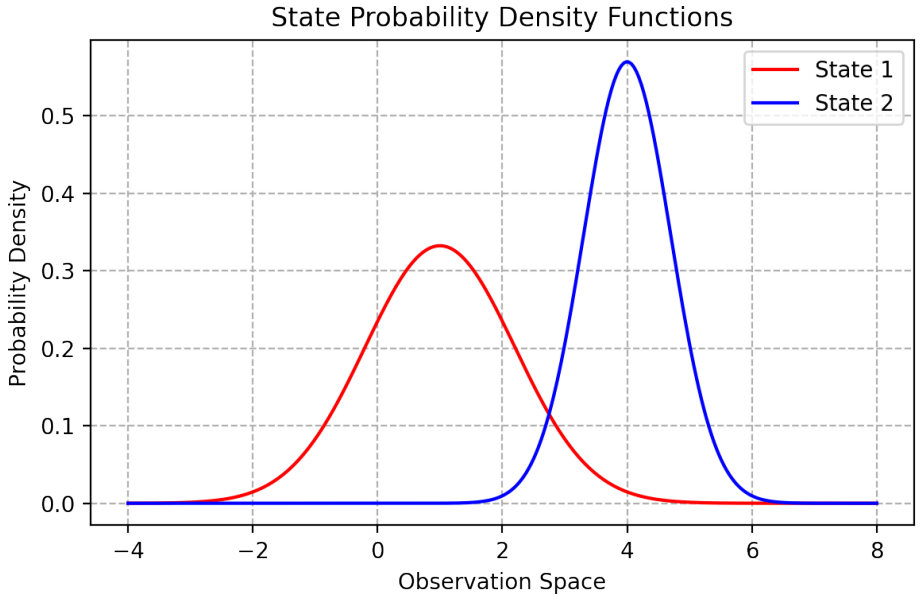

Figure 8: Probability density functions for both states across the observation space

t	o_t	$b_i(o_t)$	
		$i = 1$	$i = 2$
1	3.8	0.022	0.55
2	4.2	0.0095	0.55
3	3.4	0.045	0.39
4	-0.4	0.17	1.5
5	1.9	0.25	0.0063
6	3.0	0.083	0.21
7	1.6	0.29	0.0016
8	1.9	0.25	0.0063
9	5.0	0.0013	0.21

Table 1: Probability densities for each observation from each state (2 s.f)

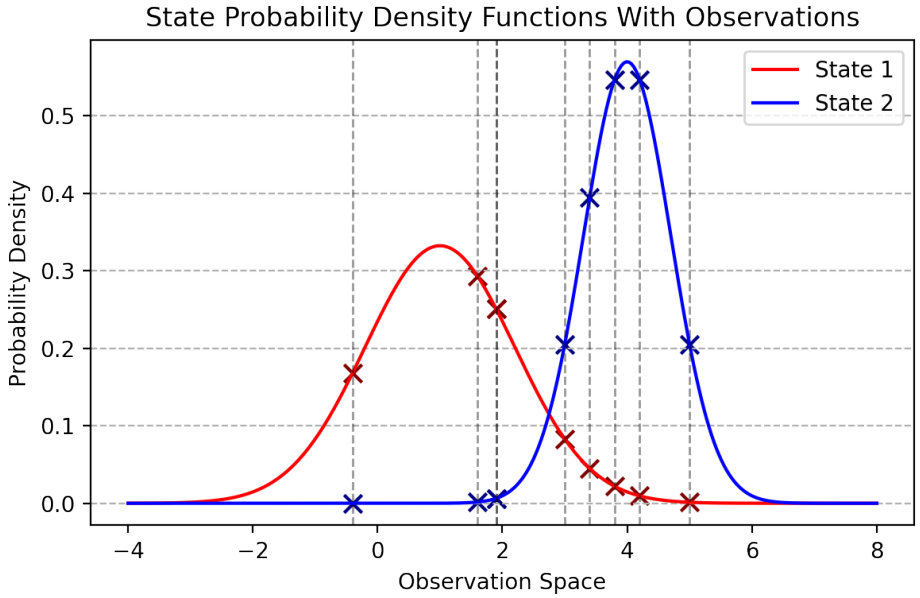

Figure 9: Previous PDFs (figure 8) with observations highlighted

3.2 Observation Probability Densities

The output probability densities for each observation in each state can be seen in table 1. These observations can be seen overlaid on the original PDFs in figure 9.

3.3 Forward Procedure and Likelihoods

The forward likelihoods calculated as part of the forward procedure can be seen in figure 2. The natural logarithm of these results can be seen graphically in figure 10. The log-likelihoods were used here as subsequent original values can decrease by orders of magnitude making it hard to visualise, as a monotonic function [9], [11], the relationship between values remains relevant when presenting the log of each value.

Both states generally decrease over time, however, state 1 has a peak while state 2 drastically drops around $t = 4$. State 1 then finally peaks again at $t = 9$ to finish higher than state 2.

Using the forward procedure (figure 2), the observation likelihood, $P(O|\lambda)$, was calculated as follows,

t	$\alpha_t(i)$	
	$i = 1$	$i = 2$
1	9.61×10^{-3}	0.306
2	2.00×10^{-4}	0.156
3	2.89×10^{-4}	0.0573
4	4.31×10^{-4}	8.01×10^{-11}
5	9.95×10^{-5}	1.64×10^{-7}
6	7.59×10^{-6}	1.26×10^{-6}
7	2.06×10^{-6}	2.59×10^{-9}
8	4.76×10^{-7}	7.99×10^{-10}
9	5.63×10^{-10}	6.02×10^{-9}

Table 2: Forward likelihoods for each state at each time step

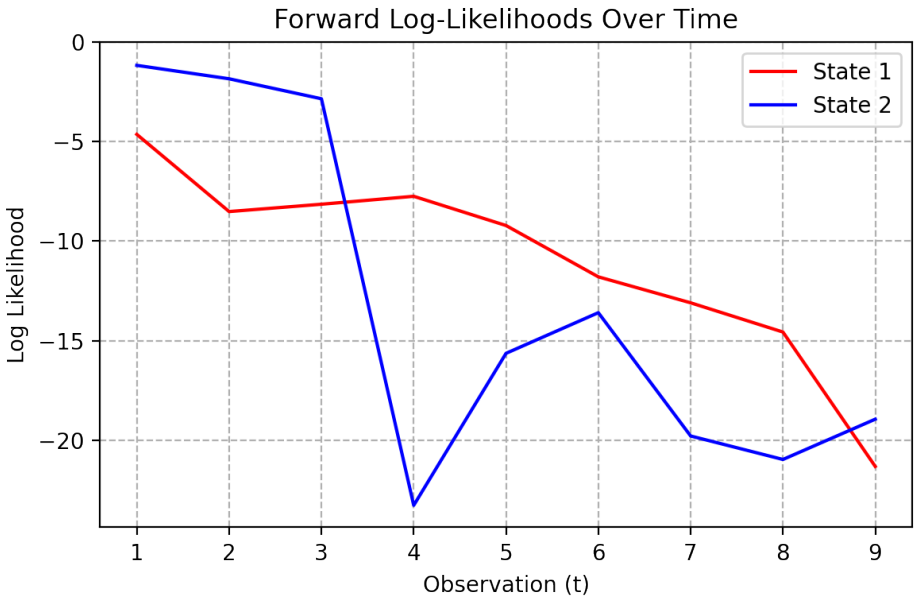

Figure 10: Forward log-likelihood for each state over all observations

t	$\beta_t(i)$	
	$i = 1$	$i = 2$
1	6.58×10^{-11}	6.25×10^{-10}
2	2.93×10^{-9}	1.23×10^{-9}
3	6.90×10^{-8}	3.00×10^{-9}
4	4.45×10^{-7}	2.11×10^{-8}
5	1.93×10^{-6}	3.02×10^{-7}
6	2.51×10^{-5}	1.15×10^{-6}
7	9.30×10^{-5}	3.77×10^{-5}
8	3.93×10^{-4}	5.73×10^{-3}
9	0.02	0.03

Table 3: Backward likelihoods for each state at each time step

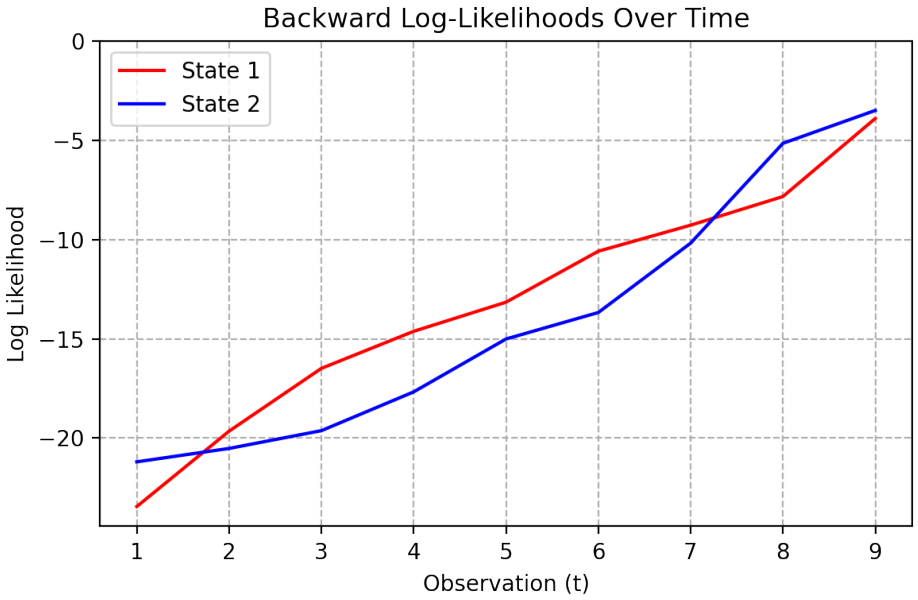

Figure 11: Backward log-likelihood for each state over all observations

$$P(O|\lambda) = \eta_1 \alpha_9(1) + \eta_2 \alpha_9(2)$$

$$P(O|\lambda) = \left(0.02 \cdot 5.63 \times 10^{-10}\right) + \left(0.03 \cdot 6.02 \times 10^{-9}\right) = 1.92 \times 10^{-10} \quad (1)$$

to 2 decimal places.

3.4 Backward Procedure and Likelihoods

The backward likelihoods were calculated as part of the backwards procedure and can be seen in table 3, the log-likelihoods can be seen in figure 11. Reading backwards in time (from bottom to top), for both states the backwards likelihood can be seen to decrease to a minimum at $t = 1$.

Finalising the backward procedure can also produce $P(O|\lambda)$,

t	$\gamma_t(i)$	
	$i = 1$	$i = 2$
1	0.003295	0.9967
2	0.003055	0.9969
3	0.1040	0.8960
4	0.9999...	8.818×10^{-9}
5	0.9997	0.0002579
6	0.9925	0.007517
7	0.9995	0.0005095
8	0.9761	0.02386
9	0.05868	0.9413

Table 4: Occupation likelihoods for each state at each time step (4 s.f.)

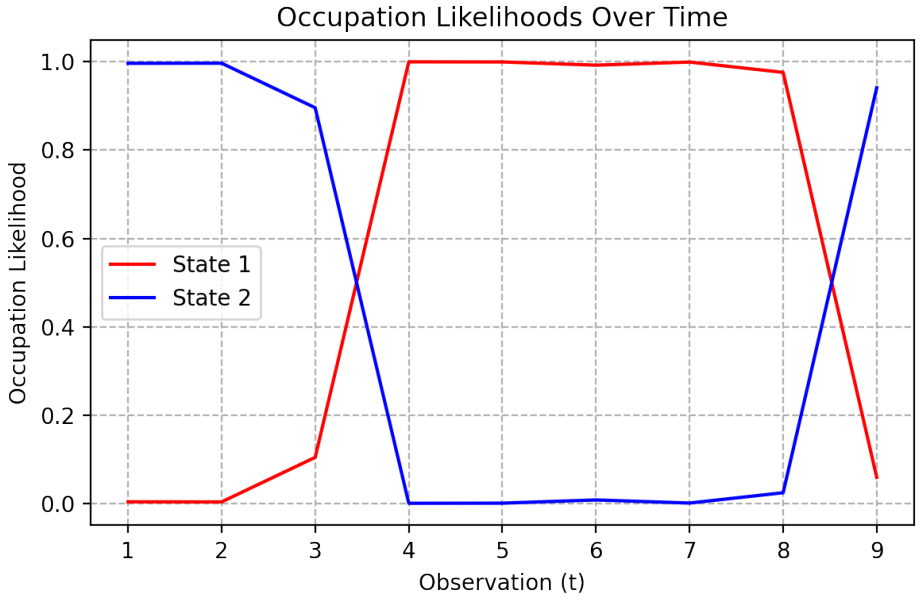

Figure 12: Occupation likelihoods for each state for each observation

$$P(O|\lambda) = \pi_1 b_1(o_1)\beta_1(1) + \pi_2 b_2(o_1)\beta_1(2)$$

$$P(O|\lambda) = \left(0.44 \cdot 0.022 \cdot 6.58 \times 10^{-11}\right) + \left(0.56 \cdot 0.55 \cdot 6.25 \times 10^{-10}\right) = 1.92 \times 10^{-10}$$

to 2 decimal places. Looking back to equation 1, these can be seen to be the same, as expected.

3.5 Occupation Likelihoods

The above forward and backward likelihoods were used to calculate the occupation likelihoods of each state at each time step, the results can be seen in table 4 and are visualised in figure 12.

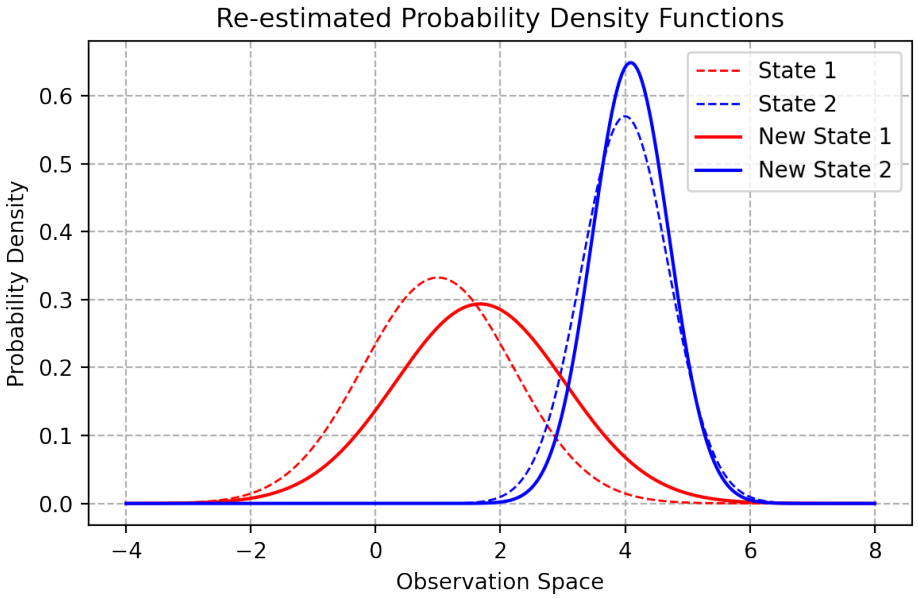

Figure 13: Probability density functions following re-estimation of parameters

3.6 Re-estimated Output Parameters

Following one iteration of Baum-Welch training, the output Gaussian parameters were re-estimated as follows (2 d.p.),

State, i	1	2
Mean, μ_i	1.67	4.09
Variance, Σ_i	1.85	0.38

This represents the following deltas from the original parameters (2 s.f.),

State, i	1	2
Mean, μ_i	+0.67	+0.089
Variance, Σ_i	+0.41	-0.11

Displayed graphically, these two sets of Gaussian functions can be seen in figure 13. Compared to the original function, increasing the standard deviation has widened and shortened the new function for state 1. The slight increase in mean has also shifted the function to the right. For state 2, the new function has a tighter, taller distribution as a result of the decreased variance; the mean has, again, slightly increased.

4 Discussion

From the occupation likelihoods, the state sequence for the training data is suggested to be as follows,

$$S = \{2, 2, 2, 1, 1, 1, 1, 1, 2\}$$

Interestingly, when comparing the occupation likelihoods to each observation's probability density (P.D), the majority are in alignment, that is, the state with the highest occupation likelihood also has the higher P.D per observation. This

is not the case for $o_6 = 3.0$ where, despite state 2 having a higher output P.D, the occupation likelihoods would in fact suggest that this was emitted from state 1. This would be a result of the model topology and relative transition likelihoods, the probability that the state alternates is low (6% and 4%).

5 Conclusion

An analysis of hidden Markov models has been demonstrated with a single iteration of Baum-Welch training providing new parameter estimations. The forward and backward procedures were demonstrated and confirmed to produce the same observation likelihood. The occupation likelihoods were calculated and used to provide a suggestion to the state sequence for the observations. Finally, new output parameters were estimated and compared to the originals.

References

[1] J. Rocca, *Introduction to markov chains*, Online, Towards Data Science, Feb. 2019. [Online]. Available: https://towardsdatascience.com/brief-introduction-to-markov-chains-2c8cab9c98ab (visited on Dec. 31, 2020).

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3 (Draft). 2020, ch. A. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/A.pdf.

[3] P. Jackson, "Hmm training," *EEEM030 - Speech & Audio Processing & Recognition*, no. K, K.13, K.14, K.15, Nov. 2020.

[4] G. Van Rossum *et al.*, *Python 3 documentation*, Online, 2008. [Online]. Available: https://docs.python.org/3/.

[5] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[6] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. development team, "Jupyter notebooks - a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Scmidt, Eds., Netherlands: IOS Press, 2016, pp. 87–90. [Online]. Available: https://eprints.soton.ac.uk/403913/.

[7] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[8] E. W. Weisstein, "Gaussian function," *MathWorldA Wolfram Web Resource*, [Online]. Available: https://mathworld.wolfram.com/GaussianFunction.html.

[9] P. Jackson, "Hmm likelihoods," *EEEM030 - Speech & Audio Processing & Recognition*, no. H, H.4, H.6, H.12, Nov. 2020.

[10] ——, "Continuous hmm," *EEEM030 - Speech & Audio Processing & Recognition*, no. N, N.8, N.12, N.13, Nov. 2020.

[11] C. Stover, "Monotonic function," *MathWorldA Wolfram Web Resource*, [Online]. Available: https://mathworld.wolfram.com/MonotonicFunction.html.

A Source Code

Listing 1: Constants file defining initial state transition probabilities and PDF parameters

```
from dataclasses import dataclass
import numpy as np
from math import sqrt


@dataclass(frozen=True)  # implements constructor among other boilerplate
class State:
    mean: float
```

```
    variance: float

    entry: float  # pi
    exit: float  # eta

    @property
    def std_dev(self):
        return sqrt(self.variance)


state1 = State(1, 1.44, 0.44, 0.02)
state2 = State(4, 0.49, 0.56, 0.03)

observations = [3.8, 4.2, 3.4, -0.4, 1.9, 3.0, 1.6, 1.9, 5.0]

a_matrix = np.array([[0.92, 0.06],
                     [0.04, 0.93]])

state_transition = np.array([[0, 0.44, 0.56, 0],
                             [0, 0.92, 0.06, 0.02],
                             [0, 0.04, 0.93, 0.03],
                             [0, 0, 0, 0]])
```

Listing 2: Maths utility file with definition for a gaussian

```
from math import sqrt, pi

import numpy as np
from numpy import exp

root_2_pi = sqrt(2. * pi) # square root is expensive, define as constant here

def gaussian(x: float, mu: float, sd: float):
    mu_pert = x - mu # mean pertubation

    coefficient = 1. / (sd * root_2_pi)

    return coefficient * exp( - (mu_pert**2)
                                        /
                                    (2.*sd**2))
```

Listing 3: Markov model object defining forward and backward procedure, occupation likelihoods, Baum-Welch equations

```
import numpy as np

from maths import gaussian

class MarkovModel:
    """Describes a single training iteration including likelihoods and reestimation
       params"""

    def __init__(self, states: list, observations: list = list(), state_transitions:
       list = list()):
        self.observations = observations
        self.state_transitions = state_transitions
        # ^ use state number not state index, is padded by entry and exit probs

        self.states = states

        self.forward = np.zeros((len(states), len(observations)))
        self.p_obs_forward = 0

        self.backward = np.zeros((len(states), len(observations)))
        self.p_obs_backward = 0

        self.occupation = np.zeros((len(states), len(observations)))

    def get_other_state_index(self, state_in):
        """For when state changes, get other index for retrieving state transitions (
           FOR 0 INDEXING)"""
        if state_in == 0:
            return 1
        elif state_in == 1:
            return 0
        else:
            print(f"invalid␣state␣index␣provided,␣({state_in})")

    def get_other_state_number(self, state_in):
        """For when state changes, get other number for retrieving state transitions
           (FOR 1 INDEXING)"""
        return self.get_other_state_index(state_in - 1) + 1

    def populate(self):
        """Calculate all likelihoods and both P(O|model)'s"""

        self.populate_forward()
```

```
        self.calculate_p_obs_forward()
        self.populate_backward()
        self.calculate_p_obs_backward()
        self.populate_occupation()
        return self

    @property
    def observation_likelihood(self):
        """abstraction for getting P(O|model) for future calculations (occupation/
            transition)"""
        return self.p_obs_forward

    ####################################
    #           Likelihoods
    ####################################

    def populate_forward(self):
        """Populate forward likelihoods for all states/times"""

        for t, observation in enumerate(self.observations):
            # iterate through observations (time)

            for state_index, state in enumerate(self.states):
                # both states at each step

                state_number = state_index + 1
                # ^ for easier reading (arrays 0-indexed, _number 1-indexed)

                if t == 0: # calcualte initial, 0 = first row = initial
                    self.forward[state_index, t] = self.state_transitions[0,
                        state_number] * gaussian(observation, state.mean, state.
                        std_dev)
                else:
                    # each state for each time has two paths leading to it,
                    # the same state (this) and the other state (other)

                    other_index = self.get_other_state_index(state_index)
                    other_number = other_index + 1 # for 1 indexing

                    # previous value * prob of changing from previous state to
                        current
                    this_to_this = self.forward[state_index, t - 1] * self.
                        state_transitions[state_number, state_number]
                    other_to_this = self.forward[other_index, t - 1] * self.
                        state_transitions[other_number, state_number]
```

```
                self.forward[state_index, t] = (this_to_this + other_to_this) *
                    gaussian(observation, state.mean, state.std_dev)

        return self.forward

    def calculate_p_obs_forward(self):
        """Calculate, store and return P(O|model) going forwards"""

        sum = 0
        for state_index, final_likelihood in enumerate(self.forward[:, -1]):
            sum += final_likelihood * self.state_transitions[state_index + 1, -1]
            # get exit prob from state transitions ^

        self.p_obs_forward = sum
        return sum

    def populate_backward(self):
        """Populate backward likelihoods for all states/times"""

        # initialise with exit probabilities
        self.backward[:, -1] = self.state_transitions[1:len(self.states) + 1, -1]

        # below iterator skips first observation
        # (will be used when finalising P(O|model))
        # iterate backwards through observations (time) [::-1] <- reverses list
        for t, observation in list(enumerate(self.observations[1:]))[::-1]:

            # print(t, observation)
            for state_index in range(len(self.states)):

                state_number = state_index + 1
                # ^ for easier reading (arrays 0-indexed, _number 1-indexed)

                other_index = self.get_other_state_index(state_index)
                other_number = other_index + 1 # for 1 indexing

                # observation for transitions from the same state
                this_state_gaussian = gaussian(observation, self.states[state_index].
                    mean, self.states[state_index].std_dev)
                # observation for transitions from the other state
                other_state_gaussian = gaussian(observation, self.states[other_index
                    ].mean, self.states[other_index].std_dev)

                # a * b * beta
```

```
                this_from_this = self.state_transitions[state_number, state_number] *
                    this_state_gaussian * self.backward[state_index, t + 1]
                other_from_this = self.state_transitions[state_number, other_number]
                   * other_state_gaussian * self.backward[other_index, t + 1]

                self.backward[state_index, t] = this_from_this + other_from_this

        return self.backward

    def calculate_p_obs_backward(self):
        """Calculate, store and return P(O|model) going backwards"""

        sum = 0
        for state_index, initial_likelihood in enumerate(self.backward[:, 0]):

            pi = self.state_transitions[0, state_index + 1]
            b = gaussian(self.observations[0],
                         self.states[state_index].mean,
                         self.states[state_index].std_dev)
            beta = initial_likelihood

            sum +=  pi * b * beta

        self.p_obs_backward = sum
        return sum

    def populate_occupation(self):
        """Populate occupation likelihoods for all states/times"""

        for t in range(len(self.observations)):
            # iterate through observations (time)

            for state_index in range(len(self.states)):

                forward_backward = self.forward[state_index, t] * self.backward[
                   state_index, t]
                self.occupation[state_index, t] = forward_backward / self.
                   observation_likelihood

        return self.occupation

    def transition_likelihood(self, from_index, to_index, t):
        """Get specific transition likelihood given state index either side and the
           timestep"""
        #from_index = i, from equations in the notes
```

```
        #to_index = j, from equations in the notes

        if t == 0:
            print("no transition likelihood for t == 0")

        forward = self.forward[from_index, t - 1]
        transition = self.state_transitions[from_index + 1, to_index + 1]
        emission = gaussian(self.observations[t],
                            self.states[to_index].mean,
                            self.states[to_index].std_dev)
        backward = self.backward[to_index, t]

        return (forward * transition * emission * backward) / self.
            observation_likelihood

    ###################################
    #     Baum-Welch Re-estimations
    ###################################

    def reestimated_state_transitions(self):
        """Re-estimate state transitions using Baum-Welch training (Not on mark
            scheme)"""

        length = len(self.states)
        new_transitions = np.zeros((length, length))

        # i
        for from_index in range(length):
            # j
            for to_index in range(length):

                # numerator iterates from t = 1 (when 0 indexing, 2 in the notes)
                transition_sum = sum(self.transition_likelihood(from_index, to_index,
                    t)
                                     for t in range(1, len(self.observations)))
                occupation_sum = sum(self.occupation[from_index, t]
                                     for t in range(0, len(self.observations)))

                new_transitions[from_index, to_index] = transition_sum /
                    occupation_sum

        return new_transitions

    def reestimated_state_mean(self, state_index):
        """Re-estimate the gaussian mean for a state using occupation likelihoods,
```

```
        baum-welch"""

        numerator = 0 # sum over observations( occupation * observation )
        denominator = 0 # sum over observations( occupation )
        for t, observation in enumerate(self.observations):
            # iterate through observations (time)

            occupation_likelihood = self.occupation[state_index, t]

            numerator += occupation_likelihood * observation
            denominator += occupation_likelihood

        return numerator / denominator

    def reestimated_mean(self):
        """Get all re-estimated gaussian means using occupation likelihoods"""
        return [self.reestimated_state_mean(idx) for idx in range(len(self.states))]


    def reestimated_state_variance(self, state_index):
        """Re-estimate the gaussian variance for a state using occupation likelihoods
            , baum-welch"""

        numerator = 0 # sum over observations( occupation * (observation - mean)^2 )
        denominator = 0 # sum over observations( occupation )
        for t, observation in enumerate(self.observations):
            # iterate through observations (time)

            occupation_likelihood = self.occupation[state_index, t]

            numerator += occupation_likelihood * pow(observation - self.states[
                state_index].mean, 2)
            denominator += occupation_likelihood

        return numerator / denominator

    def reestimated_variance(self):
        """Get all re-estimated gaussian variances using occupation likelihoods"""
        return [self.reestimated_state_variance(idx) for idx in range(len(self.states
            ))]
```

The development of the model behind this report was completed using Jupyter Notebook. The used notebook can be seen formatted in plain text below, the relevant mark scheme elements are referenced in brackets (%%= cell delimiter).

Listing 4: Plain output of Jupyter Notebook used for development and debugging

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'
# %%
#IMPORTS AND COMMON VARIABLES
import matplotlib
import matplotlib.cm as cm
import matplotlib.pyplot as plt
from matplotlib.pyplot import savefig
import numpy as np
from math import sqrt

from constants import *
from maths import gaussian
from markov import MarkovModel
from markovlog import LogMarkovModel

fig_dpi = 200
fig_export = False

x = np.linspace(-4, 8, 300) # x values for figures
x_label = "Observation␣Space"
y_label = "Probability␣Density"

# %% [markdown]
# State Probability Functions (1)
# ===================

# %%
state_1_y = [gaussian(i, state1.mean, state1.std_dev) for i in x]
state_2_y = [gaussian(i, state2.mean, state2.std_dev) for i in x]

plt.plot(x, state_1_y, c='r', label="State␣1")
plt.plot(x, state_2_y, c='b', label="State␣2")

plt.legend()
plt.title("State␣Probability␣Density␣Functions")

plt.xlabel(x_label)
plt.ylabel(y_label)
plt.grid(linestyle="--")

fig = matplotlib.pyplot.gcf()
```

```
fig.set_dpi(fig_dpi)
fig.set_tight_layout(True)
if fig_export:
    savefig("report/res/pdfs.png")
plt.show()

# %% [markdown]
# Output Probability Densities (2)
# ==========

# %%
for obs in observations:
    print(f'{obs} -> State 1: {gaussian(obs, state1.mean, state1.std_dev)}, ',
                  f'State 2: {gaussian(obs, state2.mean, state2.std_dev)}')


# %%
state_1_y = [gaussian(i, state1.mean, state1.std_dev) for i in x]
state_2_y = [gaussian(i, state2.mean, state2.std_dev) for i in x]

plt.plot(x, state_1_y, c='r', label="State 1")
plt.plot(x, state_2_y, c='b', label="State 2")

plt.legend()
plt.title("State Probability Density Functions With Observations")

plt.xlabel(x_label)
plt.ylabel(y_label)
plt.grid(linestyle="--", axis='y')

state1_pd = [gaussian(i, state1.mean, state1.std_dev) for i in observations]
state2_pd = [gaussian(i, state2.mean, state2.std_dev) for i in observations]

##############################################
#             Observation Marks
##############################################

config = {
    "s": 65,
    "marker": 'x'
}

[plt.axvline(x=i, ls='--', lw=1.0, c=(0,0,0), alpha=0.4) for i in observations]
plt.scatter(observations, state1_pd, color=(0.5, 0, 0), **config)
plt.scatter(observations, state2_pd, color=(0, 0, 0.5), **config)
```

```
fig = matplotlib.pyplot.gcf()
fig.set_dpi(fig_dpi)
fig.set_tight_layout(True)
if fig_export:
    savefig("report/res/pdfs-w-obs.png")
plt.show()

# %% [markdown]
# # Forward Procedure (3)

# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition)
model.populate_forward()

print(model.forward)

forward = model.forward
model.calculate_p_obs_forward()


# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

state_x = np.arange(1, 10)

from numpy import log as ln

plt.plot(state_x, [ln(i) for i in model.forward[0, :]], c='r', label="State 1")
plt.plot(state_x, [ln(i) for i in model.forward[1, :]], c='b', label="State 2")

plt.ylim(top=0)

plt.legend()
plt.title("Forward Log-Likelihoods Over Time")

plt.xlabel("Observation (t)")
plt.ylabel("Log Likelihood")
plt.grid(linestyle="--")

fig = matplotlib.pyplot.gcf()
```

```
fig.set_dpi(fig_dpi)
fig.set_tight_layout(True)
if fig_export:
    savefig("report/res/forward-logline.png")
plt.show()

# %% [markdown]
# # Backward Procedure (4)

# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition)
model.populate_backward()

print(model.backward)

backward = model.backward
model.calculate_p_obs_backward()


# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

state_x = np.arange(1, 10)

from numpy import log as ln

plt.plot(state_x, [ln(i) for i in model.backward[0, :]], c='r', label="State 1")
plt.plot(state_x, [ln(i) for i in model.backward[1, :]], c='b', label="State 2")

plt.ylim(top=0)

plt.legend()
plt.title("Backward Log-Likelihoods Over Time")

plt.xlabel("Observation (t)")
plt.ylabel("Log Likelihood")
plt.grid(linestyle="--")

fig = matplotlib.pyplot.gcf()
fig.set_dpi(fig_dpi)
fig.set_tight_layout(True)
```

```
if fig_export:
    savefig("report/res/backward-logline.png")
plt.show()

# %% [markdown]
# # Compare Forward/Backward Final

# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition)
model.populate_forward()
model.populate_backward()

print("forward:", model.calculate_p_obs_forward())
print("backward:", model.calculate_p_obs_backward())

print("diff: ", model.p_obs_forward - model.p_obs_backward)

# %% [markdown]
# # Occupation Likelihoods (5)

# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

occupation = model.occupation
print(model.occupation)


# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

fig = plt.figure(figsize=(6,6), dpi=fig_dpi, tight_layout=True)
ax = fig.add_subplot(1, 1, 1, projection="3d", xmargin=0, ymargin=0)

y_width = 0.3

X = np.arange(1, 10) - 0.5
Y = np.arange(1, 3) - 0.5*y_width
X, Y = np.meshgrid(X, Y)
Z = np.zeros(model.forward.size)
```

```
dx = np.ones(model.forward.size)
dy = y_width * np.ones(model.forward.size)

colours = [*[(1.0, 0.1, 0.1) for i in range(9)], *[(0.2, 0.2, 1.0) for i in range(9)
    ]]
ax.bar3d(X.flatten(), Y.flatten(), Z,
         dx, dy, model.occupation.flatten(),
         color=colours, shade=True)

ax.set_yticks([1, 2])
ax.set_zlim(top=1.0)

ax.set_title("Occupation␣Likelihoods␣Over␣Time")
ax.set_xlabel("Observation")
ax.set_ylabel("State")
ax.set_zlabel("Occupation␣Likelihood")
ax.view_init(35, -72)
if fig_export:
    savefig("report/res/occupation-bars.png")
fig.show()


# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

state_x = np.arange(1, 10)

plt.plot(state_x, model.occupation[0, :], c='r', label="State␣1")
plt.plot(state_x, model.occupation[1, :], c='b', label="State␣2")

plt.legend()
plt.title("Occupation␣Likelihoods␣Over␣Time")

plt.xlabel("Observation␣(t)")
plt.ylabel("Occupation␣Likelihood")
plt.grid(linestyle="--")

fig = matplotlib.pyplot.gcf()
fig.set_dpi(fig_dpi)
fig.set_tight_layout(True)
if fig_export:
    savefig("report/res/occupation-line.png")
```

```
plt.show()

# %% [markdown]
# # Re-estimate Mean & Variance (6)

# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

print("mean: ", [state1.mean, state2.mean])
print("variance: ", [state1.variance, state2.variance])
print()

print("mean: ", model.reestimated_mean())
print("variance: ", model.reestimated_variance())

# %% [markdown]
# New PDFs (7)
# ====================

# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

new_mean = model.reestimated_mean()
new_var = model.reestimated_variance()
new_std_dev = [sqrt(x) for x in new_var]

state_1_y = [gaussian(i, new_mean[0], new_std_dev[0]) for i in x]
state_2_y = [gaussian(i, new_mean[1], new_std_dev[1]) for i in x]

plt.plot(x, state_1_y, c='r', label="State 1")
plt.plot(x, state_2_y, c='b', label="State 2")

plt.legend()
plt.title("Re-estimated Probability Density Functions")

plt.xlabel(x_label)
plt.ylabel(y_label)
plt.grid(linestyle="--")

fig = matplotlib.pyplot.gcf()
fig.set_dpi(fig_dpi)
```

```
plt.show()

# %% [markdown]
# # Compare PDFs (7)

# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

new_mean = model.reestimated_mean()
new_var = model.reestimated_variance()
new_std_dev = [sqrt(x) for x in new_var]

########################################
#              Original
########################################
state_1_y = [gaussian(i, state1.mean, state1.std_dev) for i in x]
state_2_y = [gaussian(i, state2.mean, state2.std_dev) for i in x]
plt.plot(x, state_1_y, '--', c='r', label="State 1", linewidth=1.0)
plt.plot(x, state_2_y, '--', c='b', label="State 2", linewidth=1.0)

########################################
#            Re-Estimated
########################################
state_1_new_y = [gaussian(i, new_mean[0], new_std_dev[0]) for i in x]
state_2_new_y = [gaussian(i, new_mean[1], new_std_dev[1]) for i in x]
plt.plot(x, state_1_new_y, c='r', label="New State 1")
plt.plot(x, state_2_new_y, c='b', label="New State 2")

plt.legend()
plt.title("Re-estimated Probability Density Functions")

plt.xlabel(x_label)
plt.ylabel(y_label)
plt.grid(linestyle="--")

fig = matplotlib.pyplot.gcf()
fig.set_dpi(fig_dpi)
fig.set_tight_layout(True)
if fig_export:
    savefig("report/res/re-est-pdfs.png")
plt.show()

# %% [markdown]
```

```
# # Multiple Iterations

# %%
iterations = 50

fig = plt.figure(dpi=fig_dpi, tight_layout=True)
ax = fig.add_subplot(1, 1, 1, xmargin=0, ymargin=0)

iter_mean = [state1.mean, state2.mean]
iter_var = [state1.variance, state2.variance]
iter_state_transitions = state_transition

ax.plot(x, [gaussian(i, iter_mean[0], sqrt(iter_var[0])) for i in x], '--', c='r',
    linewidth=1.0)
ax.plot(x, [gaussian(i, iter_mean[1], sqrt(iter_var[1])) for i in x], '--', c='b',
    linewidth=1.0)

label1=None
label2=None

for i in range(iterations):
    iter_model = MarkovModel(states=[State(iter_mean[0], iter_var[0], state1.entry,
        state1.exit),
                                     State(iter_mean[1], iter_var[1], state2.entry,
                                         state2.exit)],
                             observations=observations,
                             state_transitions=iter_state_transitions).populate()

    # NEW PARAMETERS
    iter_mean = iter_model.reestimated_mean()
    iter_var = iter_model.reestimated_variance()
    iter_state_transitions[1:3, 1:3] = iter_model.reestimated_state_transitions()

    print(f"mean ({i}): ", iter_mean)
    print(f"var ({i}): ", iter_var)
    print(iter_model.reestimated_state_transitions())
    print()

    state_1_y = [gaussian(i, iter_mean[0], sqrt(iter_var[0])) for i in x]
    state_2_y = [gaussian(i, iter_mean[1], sqrt(iter_var[1])) for i in x]

    style = '--'
    linewidth = 1.0
    if i == iterations - 1:
        style = '-'
```

```
        linewidth = 2.0
        label1='State 1'
        label2='State 2'

    ax.plot(x, state_1_y, style, c='r', label=label1, linewidth=linewidth)
    ax.plot(x, state_2_y, style, c='b', label=label2, linewidth=linewidth)

ax.set_title("Probability Density Function Iterations")

ax.set_xlabel(x_label)
ax.set_ylabel(y_label)
ax.grid(linestyle="--")
ax.legend()

if fig_export:
    savefig("report/res/iterated-pdfs.png")
fig.show()

# %% [markdown]
# # Baum-Welch State Transition Re-estimations

# %%
model = MarkovModel(states=[state1, state2],
                    observations=observations,
                    state_transitions=state_transition).populate()

print(a_matrix)
print(model.reestimated_state_transitions())
model.reestimated_state_transitions()


# %%
```

B Extensions

Figure 14 shows how the output probability density functions move during 50 iterations of training. State 1's output function can be seen to spread out (increasing variance) and shift slightly to the right (increasing mean). State 2's mean begins by increasing before decreasing below the initial value. The variance also decreases, tightening the function.

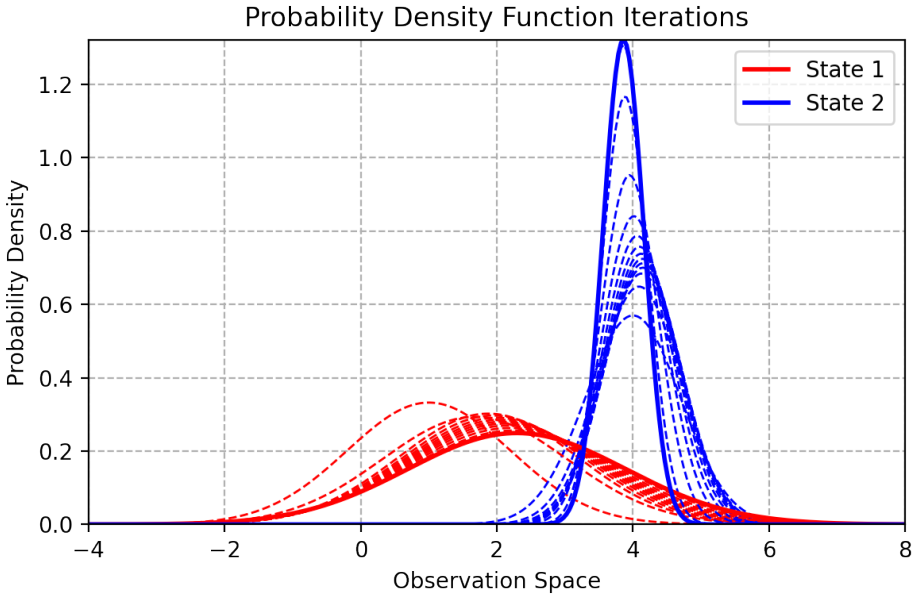

Figure 14: 5 iterations of training completed using the Baum-Welch equations