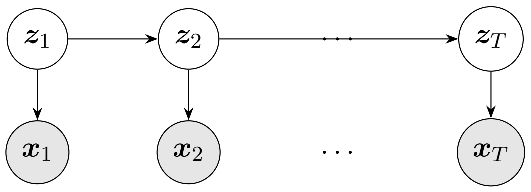
CS 1810 Spring 2025 Section 9 Notes: HMMs and MDPs

1 Hidden Markov Models

A Hidden Markov Model (HMM) is useful for inferring a sequence of unknown or hidden states from a corresponding sequence of observed evidence.

1.1 Graphical Model and Properties



We have a process that can be in one of K possible *states*, $C_1, \dots, C_K$, at each timestep. The state at a given timestep t is captured by the latent variable $\boldsymbol{z}_t$. States cannot be observed, but they generate *observations* $\boldsymbol{x}_t$ which we can see. There are M possible values, $O_1, \dots, O_M$, that each observation can take on. The state the model is in at time $t+1$ only depends on the state it was in at time t. A full HMM chain consists of T observations. Note that we then define a dataset in the context of HMM's as a collection of N of these chains, so that the dataset is written as $\{\boldsymbol{x}^{(n)}\}_{n=1}^N$ where $\boldsymbol{x}^{(n)} = (x_1^{(n)}, \dots, x_T^{(n)})$.

The key properties of HMMs are listed below.

- (*Markov property*) The next hidden state depends only on the current hidden state and nothing else.

$$p(\boldsymbol{z}_{t+1} \mid \boldsymbol{z}_1, \dots \boldsymbol{z}_t, \boldsymbol{x}_1, \dots, \boldsymbol{x}_t) = p(\boldsymbol{z}_{t+1} \mid \boldsymbol{z}_t),$$

  for $t = 1, 2, \dots T-1$.

- The current observation depends only on the current hidden state.

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_1, \dots, \boldsymbol{z}_t, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{t-1}) = p(\boldsymbol{x}_t \mid \boldsymbol{z}_t),$$

  for $t = 1, 2, \dots T$.

Observe that we can read these properties off of the graphical model!

1.2 Exercise: When to Use HMMs (Source: CMU)

For each of the following scenarios, is it appropriate to use a Hidden Markov Model? Why or why not? What would the observed data be in each case, and what would the hidden states capture?

1. Stock market price data
2. Recommendations on a database of movie reviews
3. Daily precipitation data in Boston
4. Optical character recognition for identifying words

1.3 Parameterization

We can use the HMM assumptions from above to decompose the joint distribution of the hidden states and observations into a more useful form. First, note that

$$p(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T) = p(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T) p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_T \mid \boldsymbol{z}_1, \dots, \boldsymbol{z}_T)$$

Now for the left term on the RHS, we have

$$\begin{aligned} p(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T) &= p(\boldsymbol{z}_1, \dots, \boldsymbol{z}_{T-1}) p(\boldsymbol{z}_T | \boldsymbol{z}_1, \dots, \boldsymbol{z}_{T-1}) \\ &= p(\boldsymbol{z}_1, \dots, \boldsymbol{z}_{T-1}) p(\boldsymbol{z}_T | \boldsymbol{z}_{T-1}) \\ &\vdots \\ &= p(\boldsymbol{z}_1) \prod_{t=1}^{T-1} p(\boldsymbol{z}_{t+1} \mid \boldsymbol{z}_t) \end{aligned}$$

As for the right term, we have

$$\begin{aligned} p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_T \mid \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) &= p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T-1} \mid \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) p(\boldsymbol{x}_T \mid \boldsymbol{z}_1, \dots, \boldsymbol{z}_T, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{T-1}) \\ &= p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T-1} \mid \boldsymbol{z}_1, \dots, \boldsymbol{z}_{T-1}) p(\boldsymbol{x}_T \mid \boldsymbol{z}_T) \\ &\vdots \\ &= \prod_{t=1}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{z}_t) \end{aligned}$$

In the second equality we use the fact that

$$p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T-1} \mid \boldsymbol{z}_1, \dots, \boldsymbol{z}_T) = p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T-1} \mid \boldsymbol{z}_1, \dots, \boldsymbol{z}_{T-1}),$$

which is true because $\boldsymbol{z}_{T-1}$ blocks all paths between $\boldsymbol{z}_T$ and all the observations $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T-1}$. Putting these equalities together, it follows that

$$p(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T) = p(\boldsymbol{z}_1) \prod_{t=1}^{T-1} p(\boldsymbol{z}_{t+1} \mid \boldsymbol{z}_t) \prod_{t=1}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{z}_t)$$

From this decomposition, we see that we need three parameters to fully specify our HMM:

1. $\boldsymbol{\theta} \in \mathbb{R}^K$: defines the prior distribution over initial hidden states $\boldsymbol{z}_1$. This corresponds to the term $p(\boldsymbol{z}_1)$.

2. $\boldsymbol{T} \in \mathbb{R}^{K \times K}$: transition matrix containing **state transition probabilities**. Element T_{ij} is the probability of transitioning from latent state C_i to latent state C_j. This correponds to terms of the form $p(\boldsymbol{z}_{t+1} \mid \boldsymbol{z}_t)$.

3. $\boldsymbol{\pi} \in \mathbb{R}^{K \times M}$: matrix of **emission probabilities**. Element π_{kl} is the probability of observing O_l given the latent state C_k. This corresponds to terms of the form $p(\boldsymbol{x}_t | \boldsymbol{z}_t)$.

To learn these parameters, we use expectation maximization.

1.4 Using HMMs: Inference Tasks

Before we discuss how to train HMMs through EM, it helps to gain some context on the tasks that we can use a trained HMM for. Note that we often use the term *inference* in this context to mean any task where we feed some data into our model and receive an output. Here are the major tasks that we are interested in:

1. **Filtration**: $p(\boldsymbol{z}_t | \boldsymbol{x}_1, \cdots, \boldsymbol{x}_t)$. Where am I at current time t given all the observed data up to now?

2. **Smoothing**: $p(\boldsymbol{z}_t \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$. Where was I at time t given all the observed data?

3. **Prediction**: $p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$. What will I observe at the next timestep, given everything I've observed up to now?

4. **Transition**: $p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$. What transition did I make from time t to $t+1$, given all the observed data?

5. **Joint of Observations**: $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$. What is the likelihood of observing a particular trajectory?

6. **Best Path**: $\max_{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T} p(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$. What is the most likely hidden state trajectory?

Note that if we naively compute these quantities through marginalizing, this will involve very computationally expensive sums. For instance, we recognize that smoothing can be expressed as:

$$\begin{aligned} p(\boldsymbol{z}_t \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) &\propto p(\boldsymbol{z}_t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \\ &= \sum_{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{t-1}, \boldsymbol{z}_{t+1}, \ldots, \boldsymbol{z}_T} p(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \end{aligned}$$

It follows that we seek some more clever uses of *variable elimination* to decompose the above quantities. Furthermore, imagine that we then wish to perform smoothing for $t+1$. We have

$$p(\boldsymbol{z}_{t+1} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \propto \sum_{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t, \boldsymbol{z}_{t+2}, \ldots, \boldsymbol{z}_T} p(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T),$$

which shows that we would be duplicating many of the computations that we already performed for the time t smoothing. Hence, we also want to utilize some kind of *dynamic programming* so that we can compute intermediate values in these inference tasks and reuse them efficiently to form the final output.

1.5 Forward-Backward Algorithm

Our approach to performing efficient inference relies on computing intermediate quantities that we use for all of the above tasks. The **forward-backward algorithm** utilizes efficient variable elimination and DP to compute the two sets of values: $\{\alpha_t(\boldsymbol{z}_t)\}_{t=1}^T$ and $\{\beta_t(\boldsymbol{z}_t)\}_{t=1}^T$. Below are the definitions and intuitions behind these quantities:

- $\alpha_t(\boldsymbol{z}_t)$ represents the joint probability of observations $\boldsymbol{x}_1, \dots, \boldsymbol{x}_t$ and state $\boldsymbol{z}_t$:

$$\begin{aligned}
\alpha_t(\boldsymbol{z}_t) &= p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_t, \boldsymbol{z}_t) \\
&= p(\boldsymbol{x}_t | \boldsymbol{z}_t, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{t-1}) p(\boldsymbol{z}_t, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{t-1}) \\
&= p(\boldsymbol{x}_t | \boldsymbol{z}_t) \sum_{\boldsymbol{z}_{t-1}} p(\boldsymbol{z}_t, \boldsymbol{z}_{t-1}, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{t-1}) \\
&= p(\boldsymbol{x}_t | \boldsymbol{z}_t) \sum_{\boldsymbol{z}_{t-1}} p(\boldsymbol{z}_t | \boldsymbol{z}_{t-1}, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{t-1}) p(\boldsymbol{z}_{t-1}, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{t-1}) \\
&= p(\boldsymbol{x}_t \mid \boldsymbol{z}_t) \sum_{\boldsymbol{z}_{t-1}} p(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1}) \alpha_{t-1}(\boldsymbol{z}_{t-1})
\end{aligned}$$

  Thus, we have that

$$\alpha_t(\boldsymbol{z}_t) = \begin{cases} p(\boldsymbol{x}_1 \mid \boldsymbol{z}_1) p(\boldsymbol{z}_1) & \text{if } t = 1 \\ p(\boldsymbol{x}_t \mid \boldsymbol{z}_t) \sum_{\boldsymbol{z}_{t-1}} p(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1}) \alpha_{t-1}(\boldsymbol{z}_{t-1}) & \text{if } 1 < t \leq T \end{cases}$$

  Intuitively, this allows us to quickly answer the question: "how likely are we to currently be in state $\boldsymbol{z}_t$, if we observed a specific list of values?" As we see above, it turns out that α_t can be defined in terms of α_{t-1}. That is, we move **forwards** through the sequence to calculate the α's.

- $\beta_t(\boldsymbol{z}_t)$ represents the joint probability of observations $\boldsymbol{x}_{t+1}, \dots, \boldsymbol{x}_T$ conditioned on state $\boldsymbol{z}_t$:

$$\begin{aligned}
\beta_t(\boldsymbol{z}_t) &= p(\boldsymbol{x}_{t+1}, \dots, \boldsymbol{x}_T | \boldsymbol{z}_t) \\
&= \sum_{\boldsymbol{z}_{t+1}} p(\boldsymbol{x}_{t+1}, \dots, \boldsymbol{x}_T, \boldsymbol{z}_{t+1} | \boldsymbol{z}_t) \\
&= \sum_{\boldsymbol{z}_{t+1}} p(\boldsymbol{x}_{t+1} | \boldsymbol{z}_t, \boldsymbol{z}_{t+1}, \boldsymbol{x}_{t+2}, \dots, \boldsymbol{x}_T) p(\boldsymbol{z}_{t+1}, \boldsymbol{x}_{t+2}, \dots, \boldsymbol{x}_T | \boldsymbol{z}_t) \\
&= \sum_{\boldsymbol{z}_{t+1}} p(\boldsymbol{x}_{t+1} | \boldsymbol{z}_{t+1}) p(\boldsymbol{x}_{t+2}, \dots, \boldsymbol{x}_T | \boldsymbol{z}_{t+1}, \boldsymbol{z}_t) p(\boldsymbol{z}_{t+1} | \boldsymbol{z}_t) \\
&= \sum_{\boldsymbol{z}_{t+1}} p(\boldsymbol{z}_{t+1} | \boldsymbol{z}_t) p(\boldsymbol{x}_{t+1} | \boldsymbol{z}_{t+1}) \beta_{t+1}(\boldsymbol{z}_{t+1})
\end{aligned}$$

Thus, we have that

$$\beta_t(\boldsymbol{z}_t) = \begin{cases} 1 & \text{if } t = T \\ \sum_{\boldsymbol{z}_{t+1}} p(\boldsymbol{z}_{t+1} \mid \boldsymbol{z}_t) p(\boldsymbol{x}_{t+1} \mid \boldsymbol{z}_{t+1}) \beta_{t+1}(\boldsymbol{z}_{t+1}) & \text{if } 1 \leq t < T \end{cases}$$

We can think about this intuitively by asking "what are the chances of the next observations if we are currently in state $\boldsymbol{z}_t$?" It turns out that β_t can be defined in terms of β_{t+1}. That is, we move **backwards** through the sequence to calculate the β's.

After we run the forward-backward algorithm, we can perform our desired inference tasks as such:

- **Filtration**:
$$p(\boldsymbol{z}_t | \boldsymbol{x}_1, \cdots, \boldsymbol{x}_t) \propto \alpha_t(\boldsymbol{z}_t)$$
- **Smoothing**:
$$p(\boldsymbol{z}_t \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \propto \alpha_t(\boldsymbol{z}_t)\beta_t(\boldsymbol{z}_t)$$
- **Prediction**:
$$p(\boldsymbol{x}_{T+1} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \propto \sum_{\boldsymbol{z}_T, \boldsymbol{z}_{T+1}} \alpha_T(\boldsymbol{z}_T) p(\boldsymbol{z}_{T+1} \mid \boldsymbol{z}_T) p(\boldsymbol{x}_{T+1} \mid \boldsymbol{z}_{T+1})$$
- **Transition**:
$$p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \propto \alpha_t(\boldsymbol{z}_t) p(\boldsymbol{z}_{t+1} \mid \boldsymbol{z}_t) p(\boldsymbol{x}_{t+1} \mid \boldsymbol{z}_{t+1}) \beta_{t+1}(\boldsymbol{z}_{t+1})$$
- **Joint of Observations**:
$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = \sum_{\boldsymbol{z}_t} \alpha_t(\boldsymbol{z}_t)\beta_t(\boldsymbol{z}_t),$$
for any t.
- **Best Path**: The solution doesn't actually use the same α and β values, but it is related because it also uses recursion to perform a forward and backward pass. The result says
$$\max_{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T} p(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = \max_{\boldsymbol{z}_T} \gamma_T(\boldsymbol{z}_T),$$
where
$$\gamma_t(\boldsymbol{z}_t) = \begin{cases} p(\boldsymbol{x}_1 | \boldsymbol{z}_1) p(\boldsymbol{z}_1) & \text{if } t = 1 \\ p(\boldsymbol{x}_t | \boldsymbol{z}_t) \max_{\boldsymbol{z}_{t-1}} [p(\boldsymbol{z}_t | \boldsymbol{z}_{t-1}) \gamma_{t-1}(\boldsymbol{z}_{t-1})] & \text{if } 1 < t \leq T \end{cases}$$
The above is essentially a forward pass utilizing the recursive nature of γ_t. We can also find the actual argmax, i.e. the hidden states through a backward pass that utilizes the following recursion:
$$\boldsymbol{z}^*_{t-1} = \arg\max_{\boldsymbol{z}_{t-1}} [p(\boldsymbol{z}^*_t | \boldsymbol{z}_{t-1}) \gamma_{t-1}(\boldsymbol{z}_{t-1})]$$
Note that we start off with $\boldsymbol{z}^*_T = \arg\max_{\boldsymbol{z}_T} \gamma_T(\boldsymbol{z}_T)$. This overall procedure is called the **Viterbi algorithm**.

More detailed derivations of the above solutions can be found in the textbook.

1.6 Training HMMs with Expectation Maximization

Now we move on to training HMMs. It helps that we've already discussed the various inference tasks because smoothing and transition are used in EM for HMMs!

Given data points $\{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$ defined by sequences $(x_1^{(n)}, \ldots, x_T^{(n)})$ of length T represented as row vectors, we want to infer the parameters $\{\boldsymbol{T}, \boldsymbol{\theta}, \boldsymbol{\pi}\}$. Had we been given the true states, we could easily compute the joint probabilities $p(\boldsymbol{x}^{(n)}, \boldsymbol{z}^{(n)})$, write the complete-data log likelihood, and maximize with respect to the parameters. Instead, we need to estimate state distributions and parameters iteratively.

1.6.1 E-Step

Unlike the EM algorithms that we've seen up to now, we utilize two sets of proxy distributions for training HMM's. Recall that in this step we assume a fixed set of parameters $\boldsymbol{w} = \{\boldsymbol{T}, \boldsymbol{\theta}, \boldsymbol{\pi}\}$. Below are the two components of the E-step:

1. We want a proxy $\boldsymbol{q}^{(n)}$ for the distribution of the latent states $\boldsymbol{z}_1^{(n)}, \ldots, \boldsymbol{z}_T^{(n)}$. Note that $\boldsymbol{x}^{(n)}$ contains all T timesteps, so $\boldsymbol{q}^{(n)}$ is a matrix with T rows and K columns, where the rows correspond to each $\boldsymbol{z}_j$, and the columns correspond to the possible classes of the $\boldsymbol{z}_j$. The idea is to find successive approximations of this quantity based on the data we have available. We let $z_{t,k}^{(n)}$ be the indicator that $\boldsymbol{z}_t = C_k$. Then we define the t, k element of $\boldsymbol{q}$ to be

$$q_{t,k}^{(n)} = E[z_{t,k}^{(n)} | \boldsymbol{x}^{(n)}] = P(\boldsymbol{z}_t^{(n)} = C_k | \boldsymbol{x}^{(n)}).$$

   That is, this is the probability that the state at time t is in class k given all the observed emissions. Notice how this is exactly the *smoothing* quantity we had in the previous subsection, so we can compute this through applying the forward-backward algorithm to get the necessary α and β values.

2. We also want a proxy $\boldsymbol{Q}_{t,t+1}^{(n)}$ for the joint distributions of all pairs $(t, t+1)$ of consecutive states. Note that to encapsulate all possible values of the states, this would mean that $\boldsymbol{Q}_{t,t+1}^{(n)}$ is a matrix. We then define the element at row k, column l to be

$$Q_{t,t+1,k,l}^{(n)} = E[z_{t,k}^{(n)}, z_{t+1,l}^{(n)} | \boldsymbol{x}^{(n)}] = P(\boldsymbol{z}_t^{(n)} = C_k, \boldsymbol{z}_{t+1}^{(n)} = C_l | \boldsymbol{x}^{(n)}).$$

   Notice how this is exactly the *transition* quantity from earlier!

1.6.2 M-Step

Now we need to update our parameters to maximize the expected complete-data log likelihood $\mathbb{E}_{\boldsymbol{z}}[\ln p(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{w})]$. It becomes a very nasty expression, so we will not include it here. It can be found in the textbook for reference. Applying the appropriate Lagrange multipliers and maximizing with respect to each of the parameters of interest, we recover the following update equations:

$$\theta_k = \frac{\sum_{n=1}^{N} q_{1,k}^{(n)}}{N},$$

which makes intuitive sense as the sample averages of our estimated probabilities for each possible value of the initial state. Next,

$$T_{k,l} = \frac{\sum_{n=1}^{N} \sum_{t=1}^{T-1} Q_{t,t+1,k,l}^{(n)}}{\sum_{n=1}^{N} \sum_{t=1}^{T-1} q_{tk}^{(n)}},$$

which has the intuitive interpretation of the (normalized) average of the transition probabilities, and finally

$$\pi_{k,m} = \frac{\sum_{n=1}^{N} \sum_{t=1}^{T} q_{t,k}^{(n)} x_{t,m}^{(n)}}{\sum_{n=1}^{N} \sum_{t=1}^{T} q_{tk}^{(n)}},$$

which has the intuitive interpretation of a weighted average of the emissions given the state. After updating these parameters, we repeat the EM algorithm until convergence of said parameters.

1.7 Exercise: Parameter Estimation in Supervised HMMs

You are trying to predict the weather using an HMM. The hidden states are the weather of the day, which may be sunny or rainy, and the observable states are the color of the clouds, which can be white or gray. You have data on the weather and clouds from one sequence of four days (note: the hidden states are observed here):

Day	Weather	Clouds
1	Sunny	White
2	Rainy	Gray
3	Rainy	Gray
4	Sunny	Gray

1. Draw a graphical model representing the HMM.
2. Give the values of N, T, c and of the one-hot vectors $\boldsymbol{z}_1^{(1)}, \ldots, \boldsymbol{z}_4^{(1)}, \boldsymbol{x}_1^{(1)}, \ldots, \boldsymbol{x}_4^{(1)}$.
3. Estimate and interpret the values of the parameters $\boldsymbol{\theta}, \boldsymbol{T}, \{\boldsymbol{\pi}\}$ using the MLE estimators for the supervised HMM provided in the previous subsection.

1.8 Exercise: EM for HMMs

You are trying to model a toy's state using an HMM. At each time step, the toy can be active (state 1) or inactive (state 2), but you can only observe the color of the indicator light, which can be red (observation state 1) or green (observation state 2). You have collected data from one sequence:

Time	Light
1	Green
2	Red
3	Green

You initialize your EM with $\boldsymbol{\theta} = [\frac{1}{2}\ \frac{1}{2}]^\top, \boldsymbol{T} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix}, \boldsymbol{\pi}_1 = [\frac{1}{4}\ \frac{3}{4}]^\top, \boldsymbol{\pi}_2 = [\frac{3}{4}\ \frac{1}{4}]^\top$.

1. Compute $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3$ for the forward-backward algorithm using the initial parameter values.
2. Refer to the definition of $\boldsymbol{q}_t^{(1)}$ in Section 1.6.2. Now, compute the values of $\boldsymbol{q}_1^{(1)}, \boldsymbol{q}_2^{(1)}$ using the α and β values.
3. Refer to the definition of $\boldsymbol{Q}_{t,t+1}^{(1)}$ in Section 1.6.2. Compute the value of $\boldsymbol{Q}_{1,2}^{(1)}$ using the α and β values.

During EM, at one point you obtain the following values after the E step:

$$\boldsymbol{q}_1^{(1)} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \end{bmatrix}^\top, \quad \boldsymbol{q}_2^{(1)} = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \end{bmatrix}^\top, \quad \boldsymbol{q}_3^{(1)} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \end{bmatrix}^\top$$

$$\boldsymbol{Q}_{1,2}^{(1)} = \begin{bmatrix} \frac{1}{6} & \frac{1}{2} \\ \frac{1}{6} & \frac{1}{6} \end{bmatrix}, \quad \boldsymbol{Q}_{2,3}^{(1)} = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} \end{bmatrix}$$

4. Perform the M step by updating the parameters $\boldsymbol{\theta}, \boldsymbol{T}, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2$.

1.9 Exercise: Finding the Best Path

Consider an HMM with hidden states $\{\text{Rainy}, \text{Sunny}\}$ and observations $\{\text{Walk}, \text{Shop}, \text{Clean}\}$. The model parameters are

$$p(\boldsymbol{z}_1) = \begin{cases} 0.6 & \boldsymbol{z}_1 = \text{Rainy} \\ 0.4 & \boldsymbol{z}_1 = \text{Sunny} \end{cases}, \quad p(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1}) = \begin{cases} 0.7 & (\text{Rainy} \to \text{Rainy}) \\ 0.3 & (\text{Rainy} \to \text{Sunny}) \\ 0.4 & (\text{Sunny} \to \text{Rainy}) \\ 0.6 & (\text{Sunny} \to \text{Sunny}) \end{cases}$$

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_t) = \begin{cases} 0.1 & (\text{Walk} \mid \text{Rainy}) \\ 0.4 & (\text{Shop} \mid \text{Rainy}) \\ 0.5 & (\text{Clean} \mid \text{Rainy}) \\ 0.6 & (\text{Walk} \mid \text{Sunny}) \\ 0.3 & (\text{Shop} \mid \text{Sunny}) \\ 0.1 & (\text{Clean} \mid \text{Sunny}) \end{cases}$$

Given the observation sequence

$$(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) = (\text{Walk}, \text{Shop}, \text{Clean}),$$

use the Viterbi algorithm to compute the best path $(\boldsymbol{z}_1^*, \boldsymbol{z}_2^*, \boldsymbol{z}_3^*)$.

2 Markov Decision Processes

A Markov Decision Process (MDP) is a framework for modeling an agent's actions in the world. It consists of:

1. A set of states S
2. A set of actions A
3. A reward function $r : S \times A \to \mathbb{R}$
4. A transition model $p(s'|s,a)$, $\forall s, s' \in S, a \in A$.
   Note: Transitions to the next state only depend on the value of the current state (and the current action) and thus exhibit the *Markov Property*.

A *policy* π is a mapping from states to actions, i.e. $\pi : S \to A$.

2.1 Finite time horizon MDP

In the finite horizon setting, a policy may vary with the number of time periods remaining. $\pi_{(t)}$ denotes the policy with t time steps to go. T is the decision horizon. The value of a policy with t time steps to go is defined inductively to be:

$$V^{\pi}_{(t)}(s) = \begin{cases} r(s, \pi_{(1)}(s)) & \text{if } t = 1 \\ r(s, \pi_{(t)}(s)) + \sum_{s' \in S} p(s'|s, \pi_{(t)}(s)) V^{\pi}_{(t-1)}(s') & \text{o.w.} \end{cases} \tag{1}$$

The process of computing these values inductively, working from the end of the horizon to the present, is called *value iteration.* If we instead look forward in time, we are computing the expected value of the policy

$$V^{\pi}_{T}(s) = \mathbb{E}_{s_1,\ldots,s_T}\left[\sum_{t=0}^{T} r(s_t, \pi_{(T-t)}(s_t))\right] \tag{2}$$

by induction, where $s_1 := s$. $V^{\pi}(s)$ is the MDP value function.

In an MDP, the general goal is to find an optimal policy by maximizing the expected reward under the policy, i.e. maximizing the value function. This is the *planning problem.*

2.2 Infinite Horizon MDP

Policy Evaluation We can also send $T \to \infty$, i.e. have an infinite time horizon. In that case, we need a discount factor $0 < \gamma < 1$, and we want to compute the value function

$$V^\pi(s) = \mathbb{E}_{s_1,s_2,\ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))\right] \tag{3}$$

where $s_1 := s$, and the γ factor ensures convergence (assuming bounded rewards). In this setting, we only worry about stationary policies that don't vary with time. This is the *policy evaluation* problem; for any given policy π, we can find $V^\pi(s)$ by solving the system of linear equations

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^\pi(s') \tag{4}$$

These capture consistency about the value function. To solve this system, we can use Gaussian elimination, or simply iterate until convergence as in the finite horizon case.

Given a policy π and θ (small positive number), we find V^π iteratively as follows:

- Initialize: $V(s) = 0$ for all states s.
- Repeat
  - Update step:

$$V'(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V(s'),\ \forall s \tag{5}$$

  - $\Delta = \max(|V' - V|)$
  - $V \leftarrow V'$

  until $\Delta < \theta$

Value Iteration Suppose we have an optimal policy π^*. This satisfies the following set of equations known as the *Bellman equations*:

$$V^*(s) = \max_{a \in A}\left[r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s')\right] \tag{6}$$

where $V^* \triangleq V^{\pi^*}$. Assuming we know V^*, we can read off the optimal policy by setting

$$\pi^*(s) = \arg\max_{a \in A}\left[r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s')\right] \tag{7}$$

In order to find V^*, we can use *value iteration*:

- Initialize: $V(s) = 0$ for all states s.
- Update step (Bellman operator):

$$V'(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s') \right], \ \forall s \tag{8}$$

- $V \leftarrow V'$

where we iterate until convergence of V, which is guaranteed. With our converged V, we can then find π^* as in Equation 7.

Policy Iteration Another approach to planning is called *policy iteration.* To do policy iteration, we evaluate a proposed policy π by finding V^π as in Equation 4. This is Evaluation step (E step). Then, we do a policy improvement step (I step) by the equation

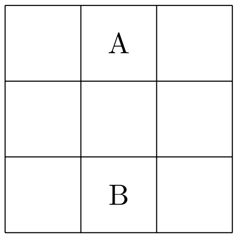
$$\pi'(s) \leftarrow \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^\pi(s') \right], \quad \forall s \tag{9}$$

We repeat the E and I steps until the policy π converges (stops changing).

Comparing Value and Policy Iteration Policy iteration involves two distinct steps: policy evaluation and policy improvement, applied iteratively to refine the policy π. Value iteration combines these steps by performing a simplified update of the value function using the Bellman optimality operator. This update implicitly improves the policy as the value function approaches optimality. Policy iteration takes more computation per iteration, but tends to converge faster in practice.

2.3 Exercise: Markov Decision Process

(Sutton & Barto 2012) Consider an MDP on the following grid:



At each square, we can go left, right, up, or down. Normally we get a reward of 0 from moving, but if we attempt to move off the grid, we get a reward of -1 and stay where we are. Also, if we move onto square A, we get a reward of 10 and are teleported to square B.

Suppose our actions also fail with probability 0.5, i.e. with probability 0.5 we stay on the current square. Also suppose our MDP is infinite horizon, and take $\gamma = 0.9$ to be the discount factor.

1. **Defining the MDP** Identify the states S, actions A, rewards, and transition probabilities $p(s'|s,a)$ in this problem.
2. **Policy Evaluation** Suppose π is the policy where we always choose to go right. Write the equations to find the values $V^\pi(s)$.
3. **Value Iteration** Write the second iteration of value iteration, i.e. starting by initializing $V(s) = \max_{a\in A} r(s,a)$.
4. **Policy Iteration** Write the first iteration of policy iteration, starting with $V^\pi(s) = 0$ for all s. (We could also initialize a policy, and do the Evaluation step to get started.)

3 Kalman Filters (Bonus Material)

Now consider the following dynamical system model:

$$z_{t+1} = \Phi z_t + \epsilon_t$$

$$x_t = A z_t + \gamma_t$$

where z are the hidden variables and x are the observed measurements. Φ and A are known constants, while ϵ and γ are random variables drawn from the following normal distributions:

$$\epsilon_t \sim \mathcal{N}(\mu_\epsilon, \sigma_\epsilon^2)$$

$$\gamma_t \sim \mathcal{N}(\mu_\gamma, \sigma_\gamma^2)$$

This is called a (one-dimensional) linear Gaussian state-space model. It is closely related to an HMM – try drawing out the graphical model! – but here the hidden states and the observations are now continuous and normally distributed. Linear Gaussian state-space models have convenient mathematical properties and can be used to describe noisy measurements of a moving object (e.g. missiles, rodents, hands), market fluctuations, etc.

The Kalman filter is an algorithm to perform filtering in linear Gaussian state-space models, i.e. to find the distribution of z_t given observations $x_1, ..., x_t$. The distribution of $z_t \,|\, x_1, ..., x_s$ will be $\mathcal{N}(\mu_{t|s}, \sigma_{t|s}^2)$. If we start with $\mu_{t-1|t-1}$ and $\sigma_{t-1|t-1}^2$, the algorithm tells us to

1. Define the distribution of $z_t \,|\, x_1, ..., x_{t-1}$ by computing $\mu_{t|t-1}$ and $\sigma_{t|t-1}^2$. This is called the prediction step.
2. Define the distribution of $z_t \,|\, x_1, ..., x_t$ by computing $\mu_{t|t}$ and $\sigma_{t|t}^2$. This is called the update step.

The Kalman filter alternates between prediction and update steps, assimilating observations one at a time. It requires one forward pass through the data, and is analogous to obtaining the α's in an HMM.