CS 181 Spring 2023 Section 9

Contents

1 Review Resources

The second midterm is coming! Our goal this week will be to provide a broad review of the content from the second half of this course, as well as to go over Markov Decision Processes (MDPs), which is our most recent content. Before diving in, we'd like to highlight a few resources that we think will be helpful for your studying:

1. Midterm Skills Checklist – This checklist is *your best guide* as to what will be covered on the midterm. (Scroll to the bottom to see the relevant parts for midterm 2 – the top is just for the last midterm!) We recommend going through it carefully once before you start studying, so that you know where your weak spots are, and going through it again at the end of your studying to make sure you haven't missed anything.

2. Midterm 2 Practice Problems There are two(!) sets of practice problems for you to go through for this midterm. We recommend spreading these out over the course of a few days and reviewing the problems that you found especially difficult.

3. Course materials: Prior section notes are a good resource for a brief refresher on content. The exercises and concept checks included in them are especially good for checking your understanding. For a more thorough review, Prof. Pan's lecture and pre-lecture notes will be very helpful.

4. OH and Sections: Both office hours and sections will be held this week. Feel free to come to office hours and ask about midterm concepts that you're confused by – they're not limited to just pset content!

2 Principal Component Analysis (PCA)

Principal component analysis (PCA) is a tool for **dimensionality reduction**. The core goal of PCA can be viewed from two perspectives:

1. Identifying and linearly projecting data onto the directions of greatest variance.
2. Minimizing the reconstruction error from returning from the lower-dimensional to the higher-dimensional (i.e. original) space.

2.1 Non-Probabilistic PCA

Suppose we have some dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, and we'd like a single principal component $v \in \mathbb{R}^D$(for simplicity's sake). Let's start with the objective of identifying and preserving the direction of greatest variance – another way of saying this is that we'd like our linear projection of $\mathbf{X}$ onto the 1D subspace determined by v to have maximal sample variance. Mathematically, we are looking for the following:

$$\begin{aligned} v^* &= \text{argmax}_v \frac{1}{N-1} \sum_{n=1}^{N} \left(v^T \mathbf{x}_n - \frac{1}{N} \sum_{i=1}^{N} v^T \mathbf{x}_i \right)^2 \\ &= \text{argmax}_v \frac{1}{N-1} \sum_{n=1}^{N} (v^T \mathbf{x}_n - v^T \bar{\mathbf{x}})^2 \end{aligned}$$

such that v is a unit vector. In class, we saw that solving this constrained optimization problem reduces to finding the eigenvector v with largest eigenvalue of the empirical covariance matrix S.

Now let's take the other perspective, where our goal is to minimize the reconstruction error when returning from the lower-dimensional space defined by v to the higher-dimensional space. We can start by observing that the reconstruction of a single point $\mathbf{x}_n$ from principal component v is:

$$\hat{\mathbf{x}}_n = v(v^T \mathbf{x}_n)$$

It's important to note that this expression for our reconstruction loss is *only* valid if our data is centered. Without data centered at the origin, no vector v will be able to reconstruct our data well. Given data that is not already centered, we can formulate our reconstruction as follows:

$$\hat{\mathbf{x}}_n = v(v^T (\mathbf{x}_n - \bar{\mathbf{x}})) + \bar{\mathbf{x}}$$

which is basically just "centering" the data and adding the mean back after projection. Given this reconstruction, "minimizing our reconstruction error" can be formalized as follows:

$$v^* = \text{argmin}_v \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{x}_n - \hat{\mathbf{x}}_n\|_2^2$$

such that v is a normal vector.

2.1.1 Non-probabilistic PCA as a Latent Variable Model

Let's again consider just a single principal component. We can consider our v to be a latent variable that determines a function mapping from a lower dimensional latent space to the target x_n that we observe. In other words, we have that $\hat{\mathbf{x}}_n = f_v(z_n) = vz_n$ and $z_n = v^T \mathbf{x}_n$. Then,

$$v^* = \text{argmin}_v \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{x}_n - f_v(z_n)\|_2^2$$

2.2 Probabilistic PCA (pPCA)

Recall our discussion of a non-probabilistic latent variable model for PCA. In going from a non-probabilistic model to a probabilistic model, we consider our observations to be *noisy* mappings from our low-dimensional latent space. In other words, we now consider the following:

$$\begin{aligned}\mathbf{z}_n &\sim \mathcal{N}(\mathbf{0}, I_{K\times K})\\ \mathbf{x}_n &= f_{\mathbf{V}}(\mathbf{z}_n) + \epsilon, \epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_{D\times D})\end{aligned}$$

where σ^2 is the strength of the noise, and we assume the latent variables are normally distributed with unit variance.

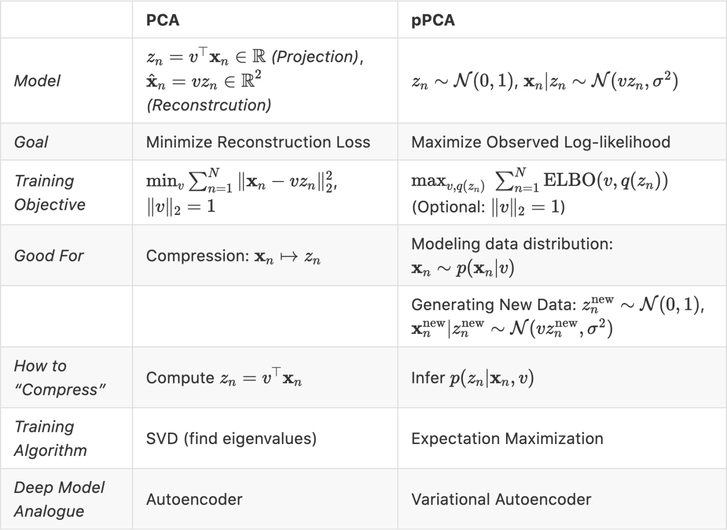
2.3 Inference for PCA

We noted previously that finding our top K principal components is equivalent to finding the K eigenvectors with largest corresponding eigenvalues from the empirical covariance matrix $\mathbf{S} = \frac{1}{N}X^TX = \frac{1}{N}\Sigma_{n=1}^{N}(x_n - \bar{x})^T(x_n - \bar{x})$, where X is the mean-centered data matrix with individual datapoints as rows. Given a matrix or linear transformation, $\mathbf{A}$, an eigenvector v of $\mathbf{A}$ is a vector that only scales, but does not change direction, when the transformation A is applied. The factor by which it scales is called the eigenvalue, λ. In other words,

$$\mathbf{A}v = \lambda v$$

We find these eigenvectors of highest corresponding eigenvalues by a method called singular value decomposition, or SVD. We recommend reviewing the Dimensionality Reduction chapter of the textbook for a brief review of SVD. For the purposes of your exam, you should understand at a high level what SVD is and how it helps us find the K largest principal components.

2.4 Synthesis



	PCA	**pPCA**
Model	$z_n = v^\top \mathbf{x}_n \in \mathbb{R}$ *(Projection)*, $\hat{\mathbf{x}}_n = vz_n \in \mathbb{R}^2$ *(Reconstrcution)*	$z_n \sim \mathcal{N}(0,1)$, $\mathbf{x}_n \vert z_n \sim \mathcal{N}(vz_n, \sigma^2)$
Goal	Minimize Reconstruction Loss	Maximize Observed Log-likelihood
Training Objective	$\min_v \sum_{n=1}^N \|\mathbf{x}_n - vz_n\|_2^2$, $\|v\|_2 = 1$	$\max_{v,q(z_n)} \sum_{n=1}^N \text{ELBO}(v, q(z_n))$ (Optional: $\|v\|_2 = 1$)
Good For	Compression: $\mathbf{x}_n \mapsto z_n$	Modeling data distribution: $\mathbf{x}_n \sim p(\mathbf{x}_n \vert v)$
		Generating New Data: $z_n^{\text{new}} \sim \mathcal{N}(0,1)$, $\mathbf{x}_n^{\text{new}} \vert z_n^{\text{new}} \sim \mathcal{N}(vz_n^{\text{new}}, \sigma^2)$
How to "Compress"	Compute $z_n = v^\top \mathbf{x}_n$	Infer $p(z_n \vert \mathbf{x}_n, v)$
Training Algorithm	SVD (find eigenvalues)	Expectation Maximization
Deep Model Analogue	Autoencoder	Variational Autoencoder

(Lecture #14 Summary)

3 Expectation Maximization (EM)

Let us consider a latent variable model $p(Y, Z|\phi, \theta)$ where Z is our unobserved (i.e. **latent**) variable and Y is our observed variable, such that our model is given by:

$$\begin{aligned} Z_n &\sim p(Z|\theta) \\ Y_n|Z_n &\sim p(Y|Z, \phi) \\ n &= 1, \ldots, N \end{aligned}$$

We'll consider first why directly maximizing the complete or observed data log-likelihood is not possible, how ELBO is used to bypass these challenges, and finally, the EM algorithm itself.

3.1 Complete vs. Observed Data Log-Likelihood

Our observed data log-likelihood is given by:

$$\log p(Y; \theta, \phi) = \sum_{n=1}^{N} \log \int_{\Omega_Z} p(y_n|z_n, \phi) p(z_n|\theta) dz$$

where Ω_Z is the domain of Z. Directly maximizing this log likelihood is made *intractable* by the integral inside the logarithm, which makes computing the gradient to solve the optimization difficult. (In the case of discrete classes Z, this analogously due to a sum inside of the log.)

But what about our complete data log-likelihood? In this case, our likelihood is:

$$\begin{aligned} \log p(Y, Z; \theta, \phi) &= \sum_{n=1}^{N} \log p(y_n, z_n|\theta, \phi) \\ &= \sum_{n=1}^{N} \log p(y_n|z_n, \phi) + \log p(z_n|\theta) \end{aligned}$$

Now, taking the gradient is not so difficult. But our complete data log-likelihood is only useful if we know Z, but we don't because it is latent! Our observation that not having to marginalize over the possible values of z_n makes optimization easier motivates our derivation of ELBO.

3.2 ELBO

We saw, in class and in section, that we can use Jensen's Inequality and the introduction of an auxiliary variable $q(Z_n)$ to derive ELBO. We'll not go through that derivation again here, but we *highly* encourage you to review this derivation from the Prof. Pan's written notes and make sure you understand why we do each step.

We define the Evidence Lower Bound (ELBO) to be:

$$\text{ELBO} = \sum_{n=1}^{N} \mathbb{E}_{z_n \sim q(Z)} \left[\log \frac{p(y_n, z_n|\theta, \phi)}{q(z_n)} \right]$$

Importantly, we found that maximizing ELBO yields a lower bound of the maximum value of the log likelihood:

$$\max_{\theta, \phi} \ell(\theta, \phi) \geq \max_{\theta, \phi, q} ELBO$$

We prefer optimizing ELBO given that the gradient is easier to evaluate:

$$\begin{aligned}\nabla_{\theta,\phi}\text{ELBO}(\theta,\phi) &= \nabla_{\theta,\phi}\left[\sum_{n=1}^{N}\mathbb{E}_{z_n\sim q(Z)}\left[\log\frac{p(y_n,z_n|\theta,\phi)}{q(z_n)}\right]\right]\\ &= \sum_{n=1}^{N}\mathbb{E}_{z_n\sim q(Z)}\left[\nabla_{\theta,\phi}\log\frac{p(y_n,z_n|\theta,\phi)}{q(z_n)}\right]\end{aligned}$$

Note that now we are able to push the gradient past the expectation, since the expectation is not computed with respect to the same variables that the gradient is.

3.3 EM Algorithm

We optimize ELBO with respect to one set of variables at a time in our EM algorithm:

STEP 1: Expectation

$$q_{\text{new}}(Z_n) = \text{argmax}_q\text{ELBO}(\theta_{\text{old}},\phi_{\text{old}}) = p(Z_n|Y_n,\theta_{\text{old}},\phi_{\text{old}})$$

STEP 2: Maximization

$$\begin{aligned}\phi_{\text{new}},\theta_{\text{new}} &= \text{argmax}_{\theta,\phi}\text{ELBO}(\theta,\phi,q_{\text{new}})\\ &= \text{argmax}_{\theta,\phi}\sum_{n=1}^{N}\mathbb{E}_{z_n\sim p(z_n|y_n,\theta_{\text{old}},\phi_{\text{old}})}\left[\log p(y_n,z_n|\theta,\phi)\right]\end{aligned}$$

Our algorithm satisfies two key properties:

1. **Monotonicity**: The expectation and maximization steps never decrease our ELBO.
2. **Convergence**: The EM algorithm will naturally converge.

3.4 Concept Check

Are the following statements true or false?

1. EM is guaranteed to converge to the global optimum of the log-likelihood.

   False. Recall that the ELBO provides a *lower-bound* on the log-likelihood, but is not equal to it.

2. EM is guaranteed to converge to the global optimum of ELBO.

   False. ELBO is generally a non-convex objective function and so EM is sensitive to initialization.

4 Example Latent Variable Models

4.1 Gaussian Mixture Models (GMMs)

Observed data, Y, is generated by a mixture of K classes, each of which has a Gaussian class-conditional distribution.

Latent: $Z_n \sim Cat(\pi)$ where $\sum_{k=1}^{K} \pi_k = 1$

Observation: $Y_n | Z_n \sim \mathcal{N}(\mu_{Z_n}, \Sigma_{Z_n})$

Example: GMMs are often used for clustering datasets such as gene expression data or customer behavior data.

4.2 Item-Response Models

A real-valued latent trait Z is measured by a set of binary observable outcomes, Y.

Latent: $Z_n \sim \mathcal{N}(\mu, \sigma^2)$ $\theta_n = g(Z_n)$ where g is some fixed function of Z_n

Observation: $Y_n | Z_n \sim Ber(\theta_n)$

Example: Item response models can be used to model the way ”underlying intelligence” relates to scores on IQ test questions.

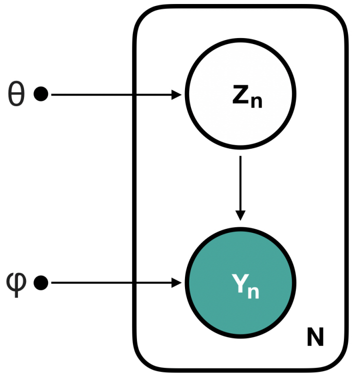
4.3 Factor Analysis Models

Observation Y with many measurements is generated by a smaller set of latent factors Z.

Latent: $Z_n \sim \mathcal{N}(0, I)$ where $Z_n \in \mathbb{R}^{D'}$

Observation: $Y_n | Z_n \sim \mathcal{N}(\mu + \Lambda Z_n, \Phi)$ where $Y_n \in \mathbb{R}^D$ and $D >> D'$

Example: Factor analysis models are especially useful for biomedical data, where harder-to-assess health factors (e.g. diabetes, cancer) generate more commonly measured characteristics like blood pressure or heart rate.



$$Z_n \sim p(Z|\theta)$$
$$Y_n|Z_n \sim p(Y|Z,\phi)$$
$$n = 1, \dots, N$$

The three latent variable models above can be described by this interaction.

5 Hidden Markov Models

5.1 Markov Chains

A Markov chain is a stochastic process with a sequence of n events $Z_1, Z_2, \ldots, Z_n$ that satisfy the **Markov Property**: conditioned on Z_{t-1}, Z_t is independent of all previous events. In other words:

$$p(Z_t | Z_1, Z_2, \ldots, Z_{t-1}) = p(Z_t | Z_{t-1})$$

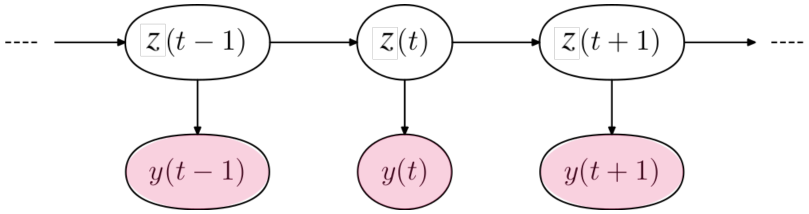
We use Markov chains to describe *sequential* or *time-series* data, such as a random walk.

If our state space is finite, then we can describe the dynamics of our chain with a **transition matrix** T, where $T_{ij} = p(Z_t = j | Z_{t-1} = i)$. If our state space is continuous, then we analogously use a **transition kernel**, where $T(z, z')$ represents the likelihood of transitioning from state $Z_{t-1} = z$ to $Z_t = z'$.

5.2 Model Parameters and Assumptions

In real-life systems, we often are not observing the true state of the system, but rather some noisy signal indicative of it. In a hidden Markov model (HMM), we no longer have access to the latent states that form a Markov chain, but rather only the observations generated by the state at each time-step. Our model makes two key assumptions:

1. The latent states must satisfy the **Markov property**.
2. Our observation y_t is dependent only on z_t, and conditioned on latent state z_t, is independent of all other states and observations.



We'll continue with sequences of length n. Our model has three sets of parameters:

1. Parameters for initial hidden state (K classes), which we'll denote as $\theta \in [0,1]^K$, where

$$p(z_t = k) = \theta_k$$
$$\sum_{k=1}^{K} \theta_k = 1$$

2. A transition matrix or kernel for transitions between states. In the discrete case, we have transition matrix $T \in [0,1]^{K \times K}$ with $T_{ij} = p(Z_t = j | Z_{t-1} = i)$.
3. The conditional probability distribution of our observation (M classes) given our state, which we'll denote $\pi \in [0,1]^{K \times M}$, where $\pi_{km} = p(y_t = m | z_t = k)$ (again, in the discrete case).

5.3 Inference

Given the parameters for our HMM, we can address a number of different kinds of queries, of which we give a few examples:

1. **Smoothing**: Given all of our observations, what do we predict from some earlier state? $p(z_t|y_1, \dots, y_n)$
2. **Transition**: Given all of our observations, what do we predict for the joint distribution of two adjacent states? $p(z_t, z_{t-1}|y_1, \dots, y_n)$
3. **Filtering**: What do we predict for our current (final) state given all taken observations? $p(z_n|y_1, \dots, y_n)$
4. **Best path/Most probable explanation**: What is the most likely series of states given all observations? $\text{argmax}_{z_{1:n}} p(z_{1:n}|y_{1:n})$

Like other latent variable models, the parameters for HMMs can be learned using EM. In this case, the expression for the ELBO of our HMM is

$$\text{ELBO} = \mathbb{E}_{z_{1:n} \sim q(Z)} \log \frac{p(z_{1:n}, y_{1:n}|\theta, \pi, T)}{q(z_{1:n})}$$

6 Topic Models

Topic modeling is a form of mixture modeling commonly used in analyzing text corpora. As this is a form of unsupervised learning, the goal is to extract a set of parameters from our data that we can use to make inferences about it. In particular, in topic modeling, we are interested in learning the topics that exist in a corpus. Remember that topic models can be defined as generative models.

6.1 Relationship with Discrete Mixtures

Let's also recap our thinking on how topic models tie to mixture models and factor analysis. The idea is in the same way in the mixture models (and factor analysis) where once when we had our z_n we could then determine our x_n (after incorporating our global parameters), in topic models once we have the π_n then we can create our x_n (after incorporating our global parameter θ).

The difference between mixture models and topic models + factor analysis, is that in topic models and factor analysis, our π and z's are continuous, whereas in mixture models, z is still discrete.

The difference between topic models and factor analysis is that we've decided in our topic models that we think the data itself (the x's) is discrete sets of counts, whereas in factor analysis our data is continuous.

6.2 Probabilistic Latent Semantic Analysis (pLSA)

The goal of pLSA is to learn the conditional probabilities $p(w \mid z)$ and $p(z \mid d)$. Given the observed data $p(w \mid d)$, we can train a latent model to estimate the conditional probabilities $p(w \mid z)$ and $p(z \mid d)$ based on the training data. The generative process for a document in pLSA is as follows:

1. Initialize conditional probabilities $p(z \mid d)$ and $p(w \mid z)$ that represent the per-document distribution for topics and per-topic distribution for words.

2. For each document d, choose a topic z with probability $p(z \mid d)$.

3. For the chosen topic z, pick a word w with probability $p(w \mid z)$.

We use the Expectation-Maximization algorithm again to maximize the likelihood of the observed data with respect to the latent variables, or the topics. The EM algorithm consists of two steps:

- Expectation (E) step: Compute the posterior probabilities of the topic assignments $p(z \mid w, d)$ using the current estimates of $p(w \mid z)$ and $p(z \mid d)$. To do so, we can use the observed data likelihood function:

$$p(w, d) = \sum_{z} p(w \mid z) p(z \mid d) p(d)$$

- Maximization (M) step: Update the estimates of $p(w \mid z)$ and $p(z \mid d)$ based on the posterior probabilities computed in the E step.

6.3 Latent Dirichlet Allocation (LDA)

We use LDA when we have to overfit and lack a generative model for new documents. Here we use a Dirichlet prior for the per-document topic distributions $\boldsymbol{\theta}$ and per-topic word distributions $\boldsymbol{\phi}$ acts as a form of regularization.We now describe the data generation process:

1. Let $\boldsymbol{\alpha} \in \mathbb{R}^{K}_{+}$ and $\boldsymbol{\beta} \in \mathbb{R}^{||}_{+}$.

2. For each document $m = 1, \ldots, M$, sample a mixture over topics: $\boldsymbol{\theta_n} \sim Dir(\boldsymbol{\alpha})$.
3. For each topic $k = 1, \ldots, K$, sample a mixture over words in that topic: $\boldsymbol{\phi_k} \sim Dir(\boldsymbol{\beta})$.
4. For each word $w_{m,n}$ (for $m = 1, \ldots, M$ and $n = 1, \ldots, N$, the length of the document), first sample the topic $z_{m,n} \sim Cat(\boldsymbol{\theta_m})$, then sample the word $w_{m,n} \sim Cat(\boldsymbol{\phi_{z_{m,n}}})$.

7 Markov Decision Processes

7.1 Definition

A Markov Decision Process is defined by the following four objects:

- S: the set of states
- A: the set of actions
- T: the transition probabilities, where $T(s'|s,a)$ is the probability of transitioning to state s' upon taking action a at state s.
- R: the reward function, where $R(s,a,s')$ is the reward of transitioning to s' upon taking action a at state s. Note: if the reward only depends on a subset of these quantities you may see it in forms such as $R(s,a)$ or even $R(s)$.

The agent may also have a discount factor $\gamma \in [0,1)$, which means that the rewards at time t are worth less by a factor of γ^t.

7.2 Goal

Our goal is to maximize the expected reward of the traveling agent. Therefore, we wish to learn some optimal policy $\pi^*(s)$ which tells us which action the agent should take at each state s. The policy can be deterministic or probabilistic, but we will only consider deterministic policies here.

If we know (S,A,T,R), then we can run an algorithm to compute the optimal policy $\pi^*(s)$. This is the **planning** task, since we can plan out π^* even before we take the first action. In the next section (reinforcement learning) the T,R is unknown so we must spend time transitioning around (exploring) the space before we can compute π^*. The rest of this section assumes that (S,A,T,R) is known and will summarize two planning algorithms (policy and value iteration) used to solve for π^*.

7.3 Definitions

1. **Cumulative Reward**: Given a series of states and actions, $\{(Z_0,A_0),(Z_1,A_1),(Z_2,A_2),\dots\}$, our **return** is

$$G = R_0 + \gamma R_1 + \gamma^2 R_2 + \dots = \sum_{n=0} \gamma^n R_n$$

2. **Value Function**: The expected return starting at state s and following policy π is

$$V^\pi(s) = \mathbb{E}_\pi[G|Z_0 = s]$$

3. **Action-Value Function**: The value of a policy, starting at state s, taking action a, and then following policy π, is

$$Q^\pi(s,a) = \mathbb{E}_\pi[G|Z_0 = s, A_0 = a]$$

7.4 Bellman Equations

In class, we derived the the Bellman Equations, which decompose V and Q into an immediate reward and the expected value of the next state. We highly encourage going through those derivations again on your own and comparing it to the derivation in class, but for simplicity we provide the equations here:

$$V^\pi(s) = \sum_{a \in A} \sum_{s' \in S} \pi(a|s) T(s'|s,a) R(s,a,s') + \gamma \sum_{a \in A} \sum_{s' \in S} \pi(a|s) T(s'|s,a) V^\pi(s')$$
$$Q^\pi(s,a) = \sum_{s' \in S} \pi(a|s) T(s'|s,a) R(s,a,s') + \gamma \sum_{s' \in S} T(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q^\pi(s',a')$$

7.5 Value Iteration

Suppose $V_t^*(s)$ is the optimal value function at s with t time steps remaining. We can solve for this by working backwards: if the agent only has one step remaining, it will take the action that has the highest reward:

$$\hat{V}_1^*(s) = \max_a R(s,a)$$

Now suppose that we have more than one time step remaining - the agent should take the action where the immediate reward plus the expected discounted value of the next state (with one less time step remaining) is maximized:

$$\widehat{V}_{t+1}^*(s) = \max_{a \in A} \left(\sum_{s' \in S} T(s'|s,a) R(s,a,s') + \gamma \sum_{s' \in S} T(s'|s,a) \widehat{V}_t^*(s') \right)$$

The chain of computations for value iteration looks like

$$\hat{V}_1^* \to \hat{V}_2^* \to \hat{V}_3^* \to \hat{V}_4^* \to \hat{V}_5^* \to \cdots$$

To find $\pi_{t+1}^*(s)$, the optimal action to take at state s with $t+1$ steps remaining, we take the argmax of the formula for $\hat{V}_{t+1}^*(s)$ instead of the max:

$$\pi_{t+1}^*(s) = \text{argmax}_a \left[\sum_{s' \in S} T(s'|s,a) R(s,a,s') + \gamma \sum_{s' \in S} T(s'|s,a) \widehat{V}_t^*(s') \right]$$

7.6 Policy iteration

In policy iteration, we repeat a two-step process: start with some initial policy $\pi_1(s)$, use it to create the corresponding value function $\hat{V}^{\pi_1}(s)$, and then use this value function to create an updated policy $\pi_2(s)$. The chain of computations looks like

$$\pi_1 \to \hat{V}^{\pi_1} \to \pi_2 \to \hat{V}^{\pi_2} \to \pi_3 \to \hat{V}^{\pi_3} \to \cdots$$

Step 1: Computing $\hat{V}^{\pi_t}(s)$ from $\pi_t(s)$ is called **policy evaluation**, where the function $V^{\pi_t}(s)$ tells us the expected value gained by following policy π_t starting from state s. We iteratively do this by starting at some guess $\hat{V}_0^{\pi_t}$, and then updating our estimate until it stops changing by more than some small positive constant:

$$\widehat{V}_{k+1}^{\pi_t}(s) = \sum_{a \in A} \sum_{s' \in S} \pi_t(a|s) T(s'|s,a) R(s,a,s') + \gamma \sum_{a \in A} \sum_{s' \in S} \pi_t(a|s) T(s'|s,a) \widehat{V}_k^{\pi_t}(s'),$$

Step 2: To find $\pi_{t+1}(s)$ given a value function $\hat{V}^{\pi_t}(s)$, we just take the action maximizes the immediate reward plus the expected future reward:

$$\begin{aligned}\pi_{t+1}(s) &= \text{argmax}_{a\in A} Q(s,a)\\ &= \text{argmax}_{a\in A}\left(\sum_{s'\in S} T(s'|s,a)R(s,a,s') + \gamma \sum_{s'\in S} T(s'|s,a)\widehat{V}^{\pi_t}(s')\right)\end{aligned}$$

In finite time, the policy will converge and reach some optimum π^*. Once this happens, the update step will not change the policy further. Since we use π_t to make $\hat{V}^{\pi_t}$, this means that the value function also converges and will not receive further updates.

7.7 Comparison

One step of value iteration takes $O(|S||A|L)$, where L is the maximum number of states reachable from any states through any action. This is intuitive, since we need to compute the value function at each s, which requires finding the expected value upon taking each action a, where the expectation is computed by summing over all the reachable states s'.

One step of policy iteration takes $O(|S||A|L + |S|^3)$, where the $|S||A|L$ comes from computing $\pi(s)$ (same reason as above) and the $|S|^3$ represents the extra work done in the policy evaluation step.

Compared to value iteration, policy iteration takes longer per iteration but takes fewer iterations to find π^*. The best to use one usually depends, but in general we prefer policy iteration.

7.8 Remark on discounting

The discount factor γ can drastically change the learned policy and value function. For a small γ, the rewards that happen in the immediate future are weighted very heavily compared to the rewards that happen later in time, so the agent is likely to be very short-sighted and optimize almost exclusively around the first few turns. For larger γ, the agent doesn't care as much about when it receives the reward, so it is willing to incur penalties in the short-term if it means that it can reap even larger rewards in the long-term.

The infinite horizon will always have a γ, but it is optional in the finite horizon case.

8 Example Problems

8.1 Expectation Maximization (#11 on practice problem set 1)

We have a collection of binary images $\mathbf{x_1}, \ldots, \mathbf{x_N}$, each of which is 5×5. We treat each image $\mathbf{x_n}$ as a 25-dimensional binary vector where the dth pixel is $x_{n,d}$. We model an image as coming from a mixture distribution, with a product-of-Bernoulli distribution for each component k:

$$p(\mathbf{x_n}; \mu_k) = \prod_{d=1}^{25} \mu_{k,d}^{x_{n,d}} (1 - \mu_{k,d})^{1-x_{n,d}} \qquad \mathbf{x_n} \in \{0,1\}^{25} \qquad \mu_k \in \{0,1\}^{25}$$

Each of the K components has parameters μ_k, where each dimension $\mu_{k,d}$ specifies the probability that pixel d is black in an example from this component. The mixture weights are $\{\theta_k\}_{k=1}^K$ and known. You will use EM to estimate the $\{\mu_k\}$ parameters.

1. Write down the probability of generating a single image $\mathbf{x}$, i.e.,

$$p(\mathbf{x}; \{\mu_k\}_{k=1}^K, \theta)$$

.

2. What are the "latent variables" in this model? Draw the plate diagram for this model, writing it for N examples, and indicating what is known and unknown. [Hint: see the previous question for a pointer to plate diagrams]

3. In the E-step, you find the probability with which example $\mathbf{x}_n$ belongs to each component fixing the parameters $\{\mu_k\}_{k=1}^K$; i.e., $q_{n,k} = p(\mathbf{z}_n = C_k | \mathbf{x}_n; \{\mu_k\}, \theta)$ for each k. Derive the expression for $q_{n,k}$.

4. In the M-step, you update the parameters $\{\mu_k\}_{k=1}^K$. Write down the expression for this, making use of the $\mathbf{q}$ values. [No need to derive the answer. As a hint, for the supervised case with class $\mathbf{z}_n$ of each image (one-hot coded), the MLE for the parameters of class k would be

$$\mu_{k,d} = \frac{\sum_{n=1}^N z_{n,k} x_{n,d}}{\sum_{n=1}^N z_{n,k}}$$

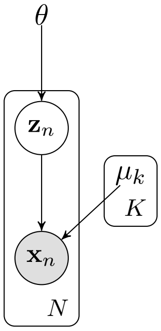
(intuitively, the percentage of times pixel d was black for the data in class k).]

Solution:

1. Sum out over possible values of the latent variables $\mathbf{z}$:

$$p(\mathbf{x}|\{\mu_k\}_{k=1}^K;\theta) = \sum_{k=1}^{K} p(\mathbf{z}=C_k;\ \theta)p(\mathbf{x}|\mathbf{z}=C_k;\mu_k) = \sum_{k=1}^{K}\theta_k \prod_{d=1}^{25} \mu_{k,d}^{x_d}(1-\mu_{k,d})^{1-x_d}$$

2. Plate diagram. The mixture weights θ and the parameters μ_k for each component k are parameters and drawn without a circle.



3. For example $\mathbf{x}_n$, we need to compute

$$q_{n,k} = p(\mathbf{z}_n = C_k|\mathbf{x}_n;\{\mu_k\},\theta)$$

We do this by applying Bayes' rule:

$$q_{n,k} = p(\mathbf{z}_n = C_k|\mathbf{x}_x;\{\mu_k\},\theta) \quad = \quad \frac{1}{Z}p(\mathbf{z}_n = C_k;\theta)p(\mathbf{x}_n|\mathbf{z}_n = C_k;\mu_k)$$

The normalization term can be computed as $Z = \sum_\ell q_{n,\ell}$.

4. For EM we instead utilize the predicted assignments of each example to each component, and have

$$\mu_{k,d} = \frac{\sum_{n=1}^N q_{n,k}x_{n,d}}{\sum_{n=1}^N q_{n,k}}$$

8.2 Alternate Reward Function for MDPs (#6 on practice problem set 1)

We have been assuming that the reward function for an MDP has the form $r(s,a)$. Also recall that we have written value iteration for infinite-horizon problems as:

$$V'(s) \leftarrow \max_{a} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V(s') \right] \tag{1}$$

Now, imagine that we have a reward function that depends on both the current state *and* the next state, i.e., $r(s,a,s')$.

1. Explain why this kind of reward function can be useful from a modeling perspective

2. Write an expression for the value iteration step that incorporates this alternative type of reward.

3. Explain formally why this approach is neither more general nor less general than an MDP model that insists on just using $r(s,a)$.

Solution.

1. This can be useful for modeling. For example, a room cleaning robot can have $r(s, pick - up, s') = 1$ if next state does not have a broken object and $r(s, pick - up, s') = -10$ if next state does have a broken object. Without this, we'd model $r(s, pick - up)$ as the expected reward for these two outcomes.

2. The value iteration step becomes:

$$V'(s) \leftarrow \max_a \left[\sum_{s'} p(s'|s,a) r(s,a,s') + \gamma \sum_{s'} p(s'|s,a) V(s') \right]$$

3. This approach can represent any $r(s, a)$ reward because we can always define $r(s, a, s') = r(s, a)$, for all values of s'. Any $r(s, a, s')$ function can be represented as an $r(s, a)$ function by defining $r(s, a) = \sum_{s'} p(s'|s, a) r(s, a, s')$.