CS 1810 Spring 2025 Final Review Session

Outline

Regression

- Linear regression and loss minimization
- Probabilistic regression and MLE
- Basis regression
- Nonparametric regression

Classification

- Logistic regression
- Gradient descent
- Generative models

Model Selection

- Bias-variance decomposition
- Ensembling and regularization
- Bayesian inference and model selection

Neural Networks

- Feedforward networks
- Backpropagation

Support Vector Machines

- Hard margin
- Soft margin
- Dual form
- Kernel trick

Clustering

- **Unsupervised learning**: learn the structure of unlabeled data. **Clustering** aims to look for groups/clusters among the data.
- For most clustering algorithms, we need a metric to specify the notion of *distance* between the data points. Common example is the l_2 distance:

$$||\mathbf{x} - \mathbf{x}'||_2 = \sqrt{\sum_{d=1}^{D} (x_d - x'_d)^2}$$

K-means

- For some choice of K and random initialization of clusters, the K-Means Algorithm (also called Lloyd's algorithm) is:
  1. For each data point, update its responsibility vector by assigning it to the cluster with the closest mean.
  $$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_{k'} ||\mathbf{x}^{(n)} - \boldsymbol{\mu}_{\boldsymbol{k'}}||_2 \\ 0 & \text{otherwise} \end{cases}$$
  2. For each cluster, update the mean $\{\boldsymbol{\mu}_{\boldsymbol{k}}\}_{k=1}^{K}$ to be the mean of the data points currently assigned to that cluster.
  $$\boldsymbol{\mu}_{\boldsymbol{k}} = \frac{\sum_{n=1}^{N} r_{nk}\boldsymbol{x}^{(n)}}{\sum_{n=1}^{N} r_{nk}}$$
  3. Repeat 1 and 2 until convergence.

Hierarchical Agglomerative Clustering (HAC)

- HAC works as follows:
  1. Start with N clusters, one for each data point.
  2. Measure the distance between clusters. This will require a **linkage criterion**.
  3. Merge the two 'closest' clusters together, reducing the number of clusters by 1. Record the distance between these two merged clusters.
  4. Repeat step 2 until we're left with only a single cluster.
- The main decision in using HAC is what the distance criterion should be between groups. A few examples are given on the next slide.

Linkage Criteria

- **Min-Linkage**:

$$\mathrm{d}_{\min}(\{\boldsymbol{x}^{(i)}\}_{i=1}^{n}, \{\boldsymbol{x}^{(i')}\}_{i'=1}^{n'}) = \min_{i,i'} ||\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i')}||_2.$$

- **Max-Linkage**:

$$\mathrm{d}_{\max}(\{\boldsymbol{x}^{(i)}\}_{i=1}^{n}, \{\boldsymbol{x}^{(i')}\}_{i'=1}^{n'}) = \max_{i,i'} ||\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i')}||_2$$

- **Average-Linkage**:

$$\mathrm{d}_{\mathrm{avg}}(\{\boldsymbol{x}^{(i)}\}_{i=1}^{n}, \{\boldsymbol{x}^{(i')}\}_{i'=1}^{n'}) = \frac{1}{nn'} \sum_{i=1}^{n} \sum_{i'=1}^{n'} ||\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i')}||_2$$

- **Centroid-Linkage**:

$$\mathrm{d}_{\mathrm{cent}}(\{\boldsymbol{x}^{(i)}\}_{i=1}^{n}, \{\boldsymbol{x}^{(i')}\}_{i'=1}^{n'}) = ||\bar{\boldsymbol{x}}_n - \bar{\boldsymbol{x}}'_{n'}||_2$$

Mixture Models

- **Mixture models** are used when you have reason to believe that each individual observation $\boldsymbol{x}^{(n)}$ has a discrete **latent variable** $\boldsymbol{z}^{(n)}$ that determines the data generating process. Latent variables = unknown, but influences the observed data.
- Let there be K possible values for each $\boldsymbol{z}^{(n)}$, denoted $\{C_k\}_{k=1}^{K}$ where each C_k is a one-hot encoded vector of length K.
- Data-generating process for each $\boldsymbol{x}^{(n)}$:
  - Sample latent class from $p(\boldsymbol{z}; \boldsymbol{\theta})$.
  - Sample from the class-conditional distribution $p(\boldsymbol{x}|\boldsymbol{z}; \boldsymbol{w})$.

Expectation Maximization: Motivation

- When training a mixture model, we have 2 goals:
  1. Compute the MLE for $\boldsymbol{w}$ and $\boldsymbol{\theta}$, i.e. the values of $\boldsymbol{w}$, $\boldsymbol{\theta}$ that maximize $p(\boldsymbol{X}; \boldsymbol{w}, \boldsymbol{\theta})$.
  2. Estimate the latent variable $\boldsymbol{z}^{(n)}$ corresponding to a particular $\boldsymbol{x}^{(n)}$, which is captured by the posterior distribution $p(\boldsymbol{z}^{(n)} | \boldsymbol{x}^{(n)}; \boldsymbol{w}, \boldsymbol{\theta})$.
- By LOTP, the log-likelihood of the observed data $\mathbf{X}$ is

$$\log p(\boldsymbol{X}; \boldsymbol{w}, \boldsymbol{\theta}) = \sum_{n=1}^{N} \log \mathbb{E}_{\boldsymbol{z}^{(n)} \sim p(\boldsymbol{z}^{(n)}; \boldsymbol{\theta})} \left[p(\boldsymbol{X}^{(n)} | \boldsymbol{z}^{(n)}; \boldsymbol{w}) \right]$$

- Two obstacles to computing a closed-form MLE:
  1. The expectation being inside the log poses a computational barrier.
  2. It is not generally true that

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p(x; \boldsymbol{\theta})} [f(x; \boldsymbol{\theta})] = \mathbb{E}_{x \sim p(x; \boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} f(x; \boldsymbol{\theta})]$$

  If the distribution of x in the above example did not depend on $\boldsymbol{\theta}$, the equality would be true.

Expectation Maximization: ELBO

- **Expectation maximization** (EM) is an approximate iterative method for MLE. Instead of maximizing likelihood, we instead maximize the **ELBO**. The term is defined as

$$ELBO = \sum_{n=1}^{N} \mathbb{E}_{\mathbf{z}^{(n)} \sim q^{(n)}(\mathbf{z}^{(n)})} \left[\log p(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}; \mathbf{w}, \boldsymbol{\theta}) - \log q^{(n)}(\mathbf{z}^{(n)}) \right]$$

- The idea of ELBO is to address obstacle 1 through using Jensen's inequality to swap the order of the log and the expectation, and to address obstacle 2 through using a proxy distribution $q^{(n)}$ in place of $p(\mathbf{z}^{(n)}; \boldsymbol{\theta})$ in the expectation.

Expectation Maximization: Algorithm

- EM continually updates $\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{\theta}$ as such:
  1. Initialize $\boldsymbol{w}^{(0)}, \boldsymbol{\theta}^{(0)}$ randomly.
  2. *(E-step)* Use the current parameters $\boldsymbol{w}^{(t)}, \boldsymbol{\theta}^{(t)}$ to update the distribution $q^{(n)}$ for each example:

$$q^{(n)}(\boldsymbol{z}^{(n)}) = \begin{bmatrix} p(\boldsymbol{z}^{(n)} = C_1 | \boldsymbol{x}^{(n)}; \boldsymbol{w}^{(t)}, \boldsymbol{\theta}^{(t)}) \\ \vdots \\ p(\boldsymbol{z}^{(n)} = C_k | \boldsymbol{x}^{(n)}; \boldsymbol{w}^{(t)}, \boldsymbol{\theta}^{(t)}) \end{bmatrix}$$

  3. *(M-step)* Update parameters by maximizing the ELBO, based on the current distributions $\boldsymbol{q}^{(n)}$ that we just updated above:

$$\begin{aligned} \boldsymbol{w}^{(t+1)}, \boldsymbol{\theta}^{(t+1)} &= \arg\max_{\boldsymbol{w}, \boldsymbol{\theta}} ELBO \\ &= \arg\max_{\boldsymbol{w}, \boldsymbol{\theta}} \sum_{n=1}^{N} \mathbb{E}_{\boldsymbol{z}^{(n)} \sim q^{(n)}(\boldsymbol{z}^{(n)})} \left[\log p(\boldsymbol{x}^{(n)}, \boldsymbol{z}^{(n)}; \boldsymbol{w}, \boldsymbol{\theta}) \right] \end{aligned}$$

  4. Go back to step 2 until the log-likelihood estimate in step 3 converges.

Clustering and Mixture Models: Concept Checks

- What is the K-means loss function?

Clustering and Mixture Models: Concept Checks

- What is the K-means loss function?
- $$\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k||^2$$

Clustering and Mixture Models: Concept Checks

- What is the K-means loss function?
- $$\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k||^2$$
- What are some key differences between K-means and HAC?

Clustering and Mixture Models: Concept Checks

- What is the K-means loss function?
- $$\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k||^2$$
- What are some key differences between K-means and HAC?
- You have to pre-determine the number of clusters in K-means. K-means depends on its initialization.

Clustering and Mixture Models: Concept Checks

- What is the K-means loss function?
- $$\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k||^2$$
- What are some key differences between K-means and HAC?
- You have to pre-determine the number of clusters in K-means. K-means depends on its initialization.
- What is the role of the expected complete data log likelihood in EM?

Clustering and Mixture Models: Concept Checks

- What is the K-means loss function?
- $$\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k||^2$$
- What are some key differences between K-means and HAC?
- You have to pre-determine the number of clusters in K-means. K-means depends on its initialization.
- What is the role of the expected complete data log likelihood in EM?
- We maximize it in the M-step!

PCA: Overview

- **Dimensionality reduction**: can be desirable for interpretability, computational efficiency, and separating signal from noise.
- **Principal component analysis (PCA)**: a re-express our original data $\boldsymbol{X}$ in terms of a lower-dimensional basis.
- These basis vectors are the *principal components*, and they are linear combinations of the original features.
- PCA has two goals when finding principal components:
  1. Ensure the components are not redundant at all, i.e. that they are orthogonal.
  2. Ensure the components capture the directions of greatest variation in the data.

PCA: Core Takeaways

- Assume $\boldsymbol{X}$ is mean-centered. Then let $\boldsymbol{S} = \frac{1}{N}\boldsymbol{X}^\top \boldsymbol{X}$ be the empirical covariance matrix.
- PCA with K components yields the K eigenvectors of $\boldsymbol{S}$ with the largest corresponding eigenvalues. Each eigenvalue corresponds to the empirical variance in that direction.
- PCA can be equivalently derived through:
  - Maximizing the total (sum of) variance explained by the principal components.
  - Minimizing reconstruction loss:

$$\mathcal{L}\left(\{\boldsymbol{z}^{(n)}\}, \boldsymbol{V}_{1:K}\right) = \frac{1}{N}\sum_{n=1}^{N} ||\boldsymbol{x}^{(n)} - \boldsymbol{V}_{1:K}\boldsymbol{V}_{1:K}^\top \boldsymbol{x}^{(n)}||_2^2$$

PCA: Computation

- How do we find the eigenvalues? Use **singular value decomposition**. Namely, let $\boldsymbol{X} = \boldsymbol{UDV}^\top$, where $\boldsymbol{U}$ is an orthogonal $N \times N$ matrix, $\boldsymbol{D}$ is a rectangular diagonal $N \times D$ matrix, and $\boldsymbol{V}$ is a $D \times D$ orthogonal matrix.
- SVD conveniently results in each column $\boldsymbol{v}_k$ of $\boldsymbol{V}$ being an eigenvector of $\boldsymbol{X}^\top \boldsymbol{X}$ with the corresponding eigenvalue equal to d_k^2, where d_k is the k-th diagonal entry of $\boldsymbol{D}$.

PCA: Concept Checks

- Write a closed-form expression for the largest eigenvalue λ_1 of $\boldsymbol{S}$ in terms of $\boldsymbol{S}$ and $\boldsymbol{v}_1$.

PCA: Concept Checks

- Write a closed-form expression for the largest eigenvalue λ_1 of $\boldsymbol{S}$ in terms of $\boldsymbol{S}$ and $\boldsymbol{v}_1$.
- $$\lambda_1 = \boldsymbol{v}_1^\top \boldsymbol{S} \boldsymbol{v}_1$$

PCA: Concept Checks

- Write a closed-form expression for the largest eigenvalue λ_1 of $\boldsymbol{S}$ in terms of $\boldsymbol{S}$ and $\boldsymbol{v}_1$.
- $$\lambda_1 = \boldsymbol{v}_1^\top \boldsymbol{S} \boldsymbol{v}_1$$
- Why might it be a bad idea to perform PCA as a pre-processing step to supervised learning?

PCA: Concept Checks

- Write a closed-form expression for the largest eigenvalue λ_1 of $\boldsymbol{S}$ in terms of $\boldsymbol{S}$ and $\boldsymbol{v}_1$.
- $$\lambda_1 = \boldsymbol{v}_1^\top \boldsymbol{S} \boldsymbol{v}_1$$
- Why might it be a bad idea to perform PCA as a pre-processing step to supervised learning?
- PCA is agnostic to the target values, so the eigenvectors you throw away may actually be the most informative.

Topic Models

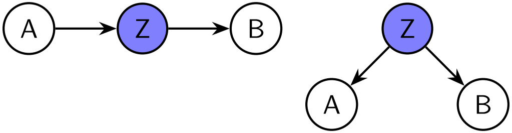
- **Topic modeling** goal: understand the underlying structure of text documents, e.g. through discovering themes.
- We have the following assumptions for our model:
  - A collection of N documents.
  - Each document is a mixture of topics, with K total possible topics.
  - We sample L total words per document.
- **Latent Dirichlet Allocation (LDA)** has the following data generation process:
  1. Let $\boldsymbol{\alpha} \in \mathbb{R}_+^K$ and $\boldsymbol{\beta} \in \mathbb{R}_+^{|\mathcal{W}|}$.
  2. For each document $n = 1, \ldots, N$, 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: $\phi_k \sim Dir(\boldsymbol{\beta})$.
  4. For each word $x_l^{(n)}$, first sample the topic $z_l^{(n)} \sim Cat(\boldsymbol{\theta}^{(n)})$, then sample the word $x_l^{(n)} \sim Cat\left(\phi_{z_l^{(n)}}\right)$.

Bayesian Networks: Overview

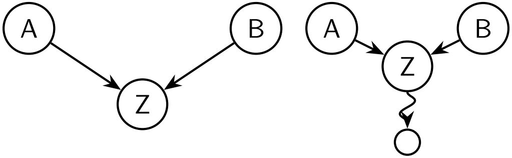
- **Bayesian networks**: represent random variables and their dependencies using a directed acyclic graph.
- Goals: reason about conditional independence and perform inference on large joint distributions.
- Let X_A and X_B denote sets of variables. An arrow $X_A \rightarrow X_B$ indicates that X_B depends on X_A. We say X_A and X_B are **d-separated** by a set of evidence X_E (observed information) if *every* undirected path from X_A to X_B is **blocked** (defined on next slide).
- If X_A and X_B are d-separated by X_E, then X_A and X_B are conditionally independent given X_E.

Bayesian Networks: d-separation

- There are two cases for a path being blocked:
  1. There is a node Z with non-converging arrows on the path, and $Z \in X_E$ (i.e., we observe Z).



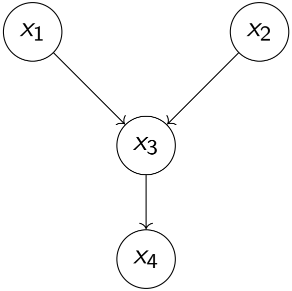
     Intuitively, observing Z blocks the flow of information from A to B.
  2. There is a node Z with converging arrows on the path, and we don't observe Z or any of its descendants.



     Explaining away: if we observe Z, this might give us information about how much A or B may have contributed to Z adopting a value.

Variable Elimination

- Consider the following network:



- Assume that all of the random variables are Categorical, with K categories.

Variable Elimination

- The joint distribution is

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)$$

- Say we want to calculate $p(x_4)$. Naive method yields:

$$p(x_4) = \sum_{x_1}\sum_{x_2}\sum_{x_3} p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)$$

- To obtain the full distribution, we need to compute this sum for $O(K)$ possible values of x_4. For each of these values of x_4, we then have to consider the K^3 possible values that (x_1, x_2, x_3) could take on. Hence, this naive calculation takes $O(K^4)$ computations.

Variable Elimination

- In this example, the optimal variable elimination procedure follows these computations:

$$\begin{aligned}
&\sum_{x_1}\sum_{x_2}\sum_{x_3} p(x_1)p(x_2)p(x_3|x_1,x_2)p(x_4|x_3)\\
&= \sum_{x_3} p(x_4|x_3)\sum_{x_2} p(x_2)\sum_{x_1} p(x_3|x_1,x_2)p(x_1)\\
&= \sum_{x_3} p(x_4|x_3)\sum_{x_2} p(x_2)p(x_3|x_2)\\
&= \sum_{x_3} p(x_4|x_3)p(x_3)\\
&= p(x_4)
\end{aligned}$$

Variable Elimination

- The elimination procedure goes from the innermost sum to the outermost sum:
  1. First, we eliminate x_1 to obtain a K by K matrix which stores the value of $p(x_3|x_2)$ for each possible pair (x_2, x_3). Since we sum over the K possible values of x_1 for each of the K^2 pairs (x_2, x_3), this step takes $O(K^3)$ computations.
  2. Eliminate x_2 and compute the K-dimensional vector $p(x_3)$. This results in $O(K^2)$ computations.
  3. Eliminate x_3 and compute the K-dimensional vector $p(x_4)$. This takes a total of $O(K^2)$ computations.
- It follows that we have performed $O(K^3) + O(K^2) + O(K^2) = O(K^3)$ total computations here, rather than the $O(K^4)$ computations we needed in the naive method.

Variable Elimination

- The problem of choosing an optimal ordering is actually NP-hard, if we don't make any assumptions on the graph structure.
- There exists an optimal strategy if the graph is a **polytree**, i.e. a directed, acyclic graph that would be a tree if we made it undirected:
  1. Prune any variables that are not ancestors of the query or evidence. Note that in an expression like $p(x|y)$, x is the query and y is the evidence.
  2. Find the variables which are furthest from the query and work backwards to perform variable elimination.

Topic Models and Graphical Models: Concept Checks

- Recall that in LDA, $\boldsymbol{\theta}^{(n)} \sim Dir(\boldsymbol{\alpha}),\ \phi_k \sim Dir(\boldsymbol{\beta})$. Are $\boldsymbol{\alpha}, \boldsymbol{\beta}$ parameters or hyperparameters?

Topic Models and Graphical Models: Concept Checks

- Recall that in LDA, $\boldsymbol{\theta}^{(n)} \sim Dir(\boldsymbol{\alpha})$, $\phi_k \sim Dir(\boldsymbol{\beta})$. Are $\boldsymbol{\alpha}, \boldsymbol{\beta}$ parameters or hyperparameters?
- Hyperparameters, as they define our priors for the distribution over topics and the distribution over words per topic.

Topic Models and Graphical Models: Concept Checks

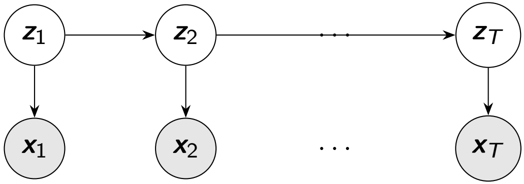
- Recall that in LDA, $\boldsymbol{\theta}^{(n)} \sim Dir(\boldsymbol{\alpha}),\ \phi_k \sim Dir(\boldsymbol{\beta})$. Are $\boldsymbol{\alpha}, \boldsymbol{\beta}$ parameters or hyperparameters?
- Hyperparameters, as they define our priors for the distribution over topics and the distribution over words per topic.
- In the algorithm for polytrees, what is the intuition behind starting far from the query and working towards it?

Topic Models and Graphical Models: Concept Checks

- Recall that in LDA, $\boldsymbol{\theta}^{(n)} \sim Dir(\boldsymbol{\alpha})$, $\phi_k \sim Dir(\boldsymbol{\beta})$. Are $\boldsymbol{\alpha}, \boldsymbol{\beta}$ parameters or hyperparameters?
- Hyperparameters, as they define our priors for the distribution over topics and the distribution over words per topic.
- In the algorithm for polytrees, what is the intuition behind starting far from the query and working towards it?
- We want to avoid eliminating variables with ancestors whenever possible because the resulting object will depend on those ancestors, leading to a higher complexity.

Hidden Markov Models: Overview

- HMMs have the following graphical model:


$$\begin{tikzcd}
\boldsymbol{z}_1 \arrow[r] \arrow[d] & \boldsymbol{z}_2 \arrow[rr, "\cdots"] \arrow[d] & & \boldsymbol{z}_T \arrow[d] \\
\boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_T
\end{tikzcd}$$

- There are K possible latent (unobserved) states $C_1, \dots, C_K$, at each timestep. State at time t is $\boldsymbol{z}_t$.
- We observe $\boldsymbol{x}_t$. There are M possible values, $O_1, \dots, O_M$, that each observation can take on.
- A full HMM chain consists of T observations. A dataset is a collection of N of these chains, so that the dataset is written as $\{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$ where $\boldsymbol{x}^{(n)} = (x_1^{(n)}, \dots, x_T^{(n)})$.

Hidden Markov Models: Key Properties

- Two key properties of HMMs:
  - (*Markov property*) The next hidden state depends only on the current hidden state and nothing else.

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

for $t = 1, 2, \dots T - 1$.
  - The current observation depends only on the current hidden state.

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

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

Hidden Markov Models: Parameterization

- Joint distribution can be decomposed:

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

- 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 $p(\boldsymbol{z}_{t+1} | \boldsymbol{z}_t)$.
  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.

Hidden Markov Models: Inference

- Here are the major inference 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**: $p(\boldsymbol{z}_t|\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$. Where was I at time t given all the observed data?
  3. **Prediction**: $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_1, \dots, \boldsymbol{x}_t)$. What will I observe at the next timestep, given everything I've observed up to now?
  4. **Transition**: $p(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}|\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$. What transition did I make from time t to $t+1$, given all the observed data?
  5. **Joint of Observations**: $p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$. What is the likelihood of observing a particular trajectory?
  6. **Best Path**: $\max_{\boldsymbol{z}_1, \dots, \boldsymbol{z}_T} p(\boldsymbol{z}_1, \dots, \boldsymbol{z}_T|\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$. What is the most likely hidden state trajectory?

Hidden Markov Models: Forward-Backward

- Naively computing the aforementioned inference tasks involves very computationally expensive sums.
- **Forward-backward algorithm**: approach to efficient inference based on dynamic programing. Idea = compute intermediate quantities used for the above tasks.
- Compute two sets of quantities
    - Forward pass: $\alpha_t(\boldsymbol{z}_t) = p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_t, \boldsymbol{z}_t)$

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

    How likely are we to currently be in state $\boldsymbol{z}_t$, if we observed a specific list of values?

    - Backward pass: $\beta_t(\boldsymbol{z}_t) = p(\boldsymbol{x}_{t+1}, \dots, \boldsymbol{x}_T|\boldsymbol{z}_t)$

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

    What are the chances of the next observations if we are currently in state $\boldsymbol{z}_t$?

Hidden Markov Models: Forward-Backward

- After we run the forward-backward algorithm, we can perform our desired inference tasks as such:
  - **Filtration**:
    $$p(\mathbf{z}_t|\mathbf{x}_1, \cdots, \mathbf{x}_t) \propto \alpha_t(\mathbf{z}_t)$$
  - **Smoothing**:
    $$p(\mathbf{z}_t|\mathbf{x}_1, \ldots, \mathbf{x}_T) \propto \alpha_t(\mathbf{z}_t)\beta_t(\mathbf{z}_t)$$
  - **Prediction**:
    $$p(\mathbf{x}_{T+1}|\mathbf{x}_1, \ldots, \mathbf{x}_T) \propto \sum_{\mathbf{z}_T, \mathbf{z}_{T+1}} \alpha_T(\mathbf{z}_T)p(\mathbf{z}_{T+1}|\mathbf{z}_T)p(\mathbf{x}_{T+1}|\mathbf{z}_{T+1})$$
  - **Transition**:
    $$p(\mathbf{z}_t, \mathbf{z}_{t+1}|\mathbf{x}_1, \ldots, \mathbf{x}_T) \propto \alpha_t(\mathbf{z}_t)p(\mathbf{z}_{t+1}|\mathbf{z}_t)p(\mathbf{x}_{t+1}|\mathbf{z}_{t+1})\beta_{t+1}(\mathbf{z}_{t+1})$$
  - **Joint of Observations**:
    $$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \sum_{\mathbf{z}_t} \alpha_t(\mathbf{z}_t)\beta_t(\mathbf{z}_t),$$
    for any t.

Hidden Markov Models: Viterbi Algorithm

- **Best Path**: The solution doesn't actually use the same α and β values, but it also performs a forward and backward pass. This is the **Viterbi algorithm**.
- 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 a forward pass utilizing the recursive nature of γ_t. We find the actual best path, i.e. the argmax, 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 $\boldsymbol{z}_T^* = \arg\max_{\boldsymbol{z}_T} \gamma_T(\boldsymbol{z}_T)$.

Hidden Markov Models: EM for Training

- 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 $\{\boldsymbol{T}, \boldsymbol{\theta}, \boldsymbol{\pi}\}$.

Hidden Markov Models: E-step

- **E-step**: Unlike the EM algorithms that we've seen up to now, we utilize two sets of proxy distributions for training HMM's:
  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. Let $z_{t,k}^{(n)}$ be the indicator that $\boldsymbol{z}_t = C_k$. Then we have
  $$q_{t,k}^{(n)} = E[z_{t,k}^{(n)} | \boldsymbol{x}^{(n)}] = P(\boldsymbol{z}_t^{(n)} = C_k | \boldsymbol{x}^{(n)}).$$
  This is exactly the *smoothing* quantity.
  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
  $$Q_{t,t+1,k,l}^{(n)} = E[z_{t,k}^{(n)}, z_{t+1,l}^{(n)} | \boldsymbol{x}^{(n)}] = P(\boldsymbol{z}_t^{(n)} = C_k, \boldsymbol{z}_{t+1}^{(n)} = C_l | \boldsymbol{x}^{(n)}).$$
  Notice that this is exactly the *transition* quantity from earlier.

Hidden Markov Models: M-step

- **M-step**: Goal is to update parameters to maximize the expected complete-data log likelihood $\mathbb{E}_{\boldsymbol{z}}[\ln p(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{w})]$.
- Derivations lead to the following update equations:
  - $$\theta_k = \frac{\sum_{n=1}^{N} q_{1,k}^{(n)}}{N},$$
    the sample average of our estimated probability of being in state C_k.
  - $$T_{k,l} = \frac{\sum_{n=1}^{N} \sum_{t=1}^{T-1} Q_{t,t+1,k,l}^{(n)}}{\sum_{n=1}^{N} \sum_{t=1}^{T-1} q_{tk}^{(n)}},$$
    the sample average of the estimated probability of transitioning from C_k to C_l.
  - $$\pi_{k,m} = \frac{\sum_{n=1}^{N} \sum_{t=1}^{T} q_{t,k}^{(n)} x_{t,m}^{(n)}}{\sum_{n=1}^{N} \sum_{t=1}^{T} q_{tk}^{(n)}},$$
    the sample average of the emission O_m given the state C_k.

Hidden Markov Models: Concept Checks

- Why do we run the forward-backward algorithm?

Hidden Markov Models: Concept Checks

- Why do we run the forward-backward algorithm?
- The resulting forward and backward values can be used for a variety of inference tasks.

Hidden Markov Models: Concept Checks

- Why do we run the forward-backward algorithm?
- The resulting forward and backward values can be used for a variety of inference tasks.
- How do we compute the proxy distributions in the E-step for training HMMs?

Hidden Markov Models: Concept Checks

- Why do we run the forward-backward algorithm?
- The resulting forward and backward values can be used for a variety of inference tasks.
- How do we compute the proxy distributions in the E-step for training HMMs?
- We use the forward and backward values, since we need to perform a smoothing task and a transition task!

Markov Decision Processes: Overview

- 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 $r : S \times A \rightarrow \mathbb{R}$
  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. $\pi : S \rightarrow A$.
- You can review **finite time horizon** MDPs in the section notes / textbook, but we will focus on the **infinite time horizon** problem.

Markov Decision Processes: Policy Evaluation

- **Policy evaluation**: We seek to compute the value function

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

  where $s_1 := s$, and $0 < \gamma < 1$ is the **discount factor** that ensures convergence.

- We can find the closed-form solution via solving the following system of linear equations:

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

- We can also find V^π iteratively:
  1. Initialize $V(s) = 0$ for all states s.
  2. Update step:

$$V'(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V(s'),\ \forall s, \quad \Delta = \max(|V' - V|),$$

  3. Repeat until $\Delta < \theta$ for some threshold θ.

Markov Decision Processes: Value Iteration

- Suppose we have an optimal policy π^*. Define $V^* \triangleq V^{\pi^*}$. This satisfies the **Bellman equations**:

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

- **Value iteration** iteratively computes V^*:
  - Initialize $V(s) = 0$ for all states s.
  - Update step (Bellman operator):

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

  - Repeat until convergence, which is guaranteed.
- Note that V^* allows us to find the optimal policy π^* because

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

Markov Decision Processes: Policy Iteration

- **Policy iteration** is another iterative approach to finding planning. It consists of an evaluation step and an improvement step.
- **Evaluation**: evaluate a proposed policy π by finding V^π.
- **Improvement**: use the equation

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

- We repeat the E and I steps until the policy π converges.
- Policy iteration takes more computation per iteration, but tends to converge faster in practice.

Reinforcement Learning: Overview

- In MDP planning, we are given the environment, i.e. the transition distribution $p(s'|s,a)$ and the reward function $r(s,a)$. In **reinforcement learning**, we are not and must interact with the environment to learn an appropriate policy.
- **Model-based learning**: We estimate the missing models, $r(s,a)$ and $p(s'|s,a)$, and then use planning (value or policy iteration) to develop a policy π.
- **Model-free learning**: We skip estimating the transition and reward functions. Instead, we directly infer the optimal policy.

Reinforcement Learning: Key Ideas

- Model-free strategy: learn the action-value function of the optimal policy, written as Q^*. We define this for all $s \in S, a \in A$ as

$$\begin{aligned} Q^*(s,a) &= r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^*(s') \\ &= r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A}[Q^*(s',a')], \end{aligned}$$

  since $V^*(s') = \max_{a' \in A}[Q^*(s',a')]$.
- Note that $\pi^*(s) = \arg\max_{a \in A} Q^*(s,a)$.
- **Exploration vs. exploitation**:
  - Exploitation: When in state s, take action $a = \arg\max_{a \in A} Q(s,a)$, which is optimal based on our current estimate of the Q-function.
  - Exploration: We want to ensure that we have visited enough states and taken enough actions from those states to get *good* Q-function estimates.

SARSA

- **SARSA (State-Action-Reward-State-Action)**: algorithm that uses the current state s, current action a, reward r, next state s', and next action a' to perform the following update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

- The term α_t, with $0 \le \alpha_t < 1$, is the learning rate at update t. γ is the discount factor. Finally, the difference term is the temporal difference (TD) error.
- An **on-policy** method because we choose a' with the **behavior policy** π that the agent follows. π is ϵ-greedy.
- SARSA, and any on-policy method in general, is not guaranteed to converge to Q^*. Conditions for convergence: (i) visit every action in every state infinitely often, (ii) decay the learning rate over time, but not too quickly, (iii) move from ϵ-greedy to greedy over time.

Q-Learning

- Q-**Learning** is similar to SARSA, but it instead uses the following update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

- An **off-policy** method because the next action used for the update is chosen greedily. Thus, it may not be the same as the actual action a' that the policy dictates the agent take at s'.
- Guaranteed to converge to Q^* as long as we: (i) visit every action in every state infinitely often, (ii) decay the learning rate over time, but not too quickly.

MDPs and RL: Concept Checks

- What is the relationship between MDPs, planning, and RL?

MDPs and RL: Concept Checks

- What is the relationship between MDPs, planning, and RL?
- MDPs provide the model for the sequential decision-making problem itself. Planning is an approach to solving MDP problems where you assume knowledge of the environment. RL is when you do not assume knowledge of the environment.

MDPs and RL: Concept Checks

- What is the relationship between MDPs, planning, and RL?
- MDPs provide the model for the sequential decision-making problem itself. Planning is an approach to solving MDP problems where you assume knowledge of the environment. RL is when you do not assume knowledge of the environment.
- How is model-based RL different from MDP planning?

MDPs and RL: Concept Checks

- What is the relationship between MDPs, planning, and RL?
- MDPs provide the model for the sequential decision-making problem itself. Planning is an approach to solving MDP problems where you assume knowledge of the environment. RL is when you do not assume knowledge of the environment.
- How is model-based RL different from MDP planning?
- In model-based RL, you have to estimate the reward and transition functions. Then you essentially just have a planning problem!