1 Derivation of PCA

Assume we are given n training data points $(\mathbf{x}_i, y_i)$. We collect the target values into $\mathbf{y} \in \mathbb{R}^n$, and the inputs into the matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ where the rows are the $d-$dimensional feature vectors $\mathbf{x}_i^\top$ corresponding to each training point. Furthermore, assume that the data has been centered such that $\frac{1}{n}\sum_{i=1}^n \mathbf{x_i} = \mathbf{0}$, $n > d$ and $\mathbf{X}$ has rank d. The covariance matrix is given by

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top$$

When $\bar{\mathbf{x}} = 0$ (i.e., we have subtracted the mean in our samples), we obtain $\Sigma = \frac{1}{n}\mathbf{X}^\top\mathbf{X}$. We will assume this to be the case for this problem.

1. Maximum Projected Variance: We would like the vector $\mathbf{w}$ such that projecting your data onto $\mathbf{w}$ will retain the maximum amount of information, i.e., variance. We can formulate the optimization problem as

$$\max_{\mathbf{w}:\|\mathbf{w}\|_2=1} \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i{}^\top\mathbf{w}\right)^2 = \max_{\mathbf{w}:\|\mathbf{w}\|_2=1} \frac{1}{n}\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}\,. \tag{1}$$

Show that the maximizer for this problem is equal to the eigenvector $\mathbf{v}_1$ that corresponds to the largest eigenvalue λ_1 of Σ. Also show that the optimal value of this problem is equal to λ_1.

Hint: Use the spectral decomposition of Σ and consider reformulating the optimization problem using a new variable.

Solution:

We are tasked with solving the following optimization problem:

$$\max_{\mathbf{w}:\|\mathbf{w}\|_2=1} \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^T\mathbf{w}\right)^2 = \max_{\mathbf{w}:\|\mathbf{w}\|_2=1} \frac{1}{n}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}$$

Firstlly, using the fact that Covariance Matrix Σ is given by: $\Sigma = \frac{1}{n}\mathbf{X}^T\mathbf{X}$. We can rewrite the optimization problem as: $\max_{\mathbf{w}:\|\mathbf{w}\|_2=1} \mathbf{w}^T\Sigma\mathbf{w}$

Secondly, since the covariance matrix Σ is symmetric and positive semi-definite, we can decompose it as: $\Sigma = \mathbf{V}\Lambda\mathbf{V}^T$

where $\mathbf{V}$ is the matrix of eigenvectors, and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$ is the diagonal matrix of eigenvalues, with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$.

Thirdly, we can express $\mathbf{w}$ as a linear combination of the eigenvectors of Σ, i.e., $\mathbf{w} = \mathbf{Va}$, where $\mathbf{a}$ is a coefficient vector. Substituting this into the objective function: $\mathbf{w}^T\Sigma\mathbf{w} = \mathbf{a}^T\mathbf{V}^T\Sigma\mathbf{Va} = \mathbf{a}^T\Lambda\mathbf{a}$. Since Λ is diagonal, this simplifies to: $\mathbf{w}^T\Sigma\mathbf{w} = \sum_{i=1}^d \lambda_i a_i^2$

Finally ,to maximize $\sum_{i=1}^d \lambda_i a_i^2$ subject to the constraint $\|\mathbf{w}\|_2 = 1$, or equivalently $\sum_{i=1}^d a_i^2 = 1$, the maximum value is achieved when all the weight is placed on the largest eigenvalue λ_1. This occurs when $a_1 = 1$ and $a_2 = a_3 = \cdots = a_d = 0$.

2. Let us call the solution of the above part $\mathbf{w}^{(1)}$. Next, we will use a *greedy procedure* to find the ith component of PCA by doing the following optimization

$$\begin{aligned} \text{maximize} \quad & \tfrac{1}{n}(\mathbf{w}^{(i)})^\top \mathbf{X}^\top \mathbf{X}\mathbf{w}^{(i)} \\ \text{subject to} \quad & \|\mathbf{w}^{(i)}\|_2 = 1 \\ & (\mathbf{w}^{(i)})^\top \mathbf{w}^{(j)} = 0 \quad \forall j < i, \end{aligned} \tag{2}$$

where the $\mathbf{w}^{(j)}$ vectors for all $j < i$ are defined recursively using the same maximization procedure above. Show, using your work in the previous part, that the maximizer for this problem is equal to the eigenvector $\mathbf{v}_i$ that corresponds to the ith eigenvalue λ_i of Σ. Also show that optimal value of this problem is equal to λ_i.

Solution:

Firstly, like the previous part, we know that the covariance matrix Σ is given by: $\Sigma = \frac{1}{n}\mathbf{X}^T\mathbf{X}$ and we rewrite the the optimization problem as: $\max_{\mathbf{w}^{(i)}} \mathbf{w}^{(i)T}\Sigma\mathbf{w}^{(i)}$ subject to $\|\mathbf{w}^{(i)}\|_2 = 1$ and $\mathbf{w}^{(i)T}\mathbf{w}^{(j)} = 0\ \forall j < i$.

Secondly, we decompose Σ as: $\Sigma = \mathbf{V}\Lambda\mathbf{V}^T$ where $\mathbf{V}$ is the matrix of eigenvectors, and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$ is the diagonal matrix of eigenvalues.

Thirdly, we rewrite the Objective Function. Let $\mathbf{w}^{(i)} = \mathbf{V}\mathbf{a}^{(i)}$, where $\mathbf{a}^{(i)}$ is a coefficient vector. Substituting this into the objective function: $\mathbf{w}^{(i)T}\Sigma\mathbf{w}^{(i)} = \mathbf{a}^{(i)T}\Lambda\mathbf{a}^{(i)} = \sum_{k=1}^{d}\lambda_k a_k^{(i)2}$

Finally, to maximize $\sum_{k=1}^{d}\lambda_k a_k^{(i)2}$ subject to $\|\mathbf{a}^{(i)}\|_2 = 1$ and the orthogonality condition $a_j^{(i)} = 0$ for all $j < i$, the maximum value is achieved when $a_i^{(i)} = 1$, and all other coefficients are zero.

2 PCA and Least Squares

Consider the ridge regression estimator,

$$\widehat{w}_{\text{ridge}} := \arg\min_{w\in\mathbb{R}^d} \|Xw - y\|_2^2 + \lambda\|w\|_2^2,$$

where $X \in \mathbb{R}^{n\times d}$ and $y \in \mathbb{R}^n$. Suppose that X has already been centered and has singular value decomposition $X = U\Sigma V^\top = \sum_{i=1}^d \sigma_i u_i v_i^\top$, where $U \in \mathbb{R}^{n\times d}$, $\Sigma \in \mathbb{R}^{d\times d}$, and $V \in \mathbb{R}^{d\times d}$, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d \geq 0$ are the diagonal components of Σ.

1. Show that

$$\widehat{w}_{\text{ridge}} = \sum_{i=1}^d \rho_\lambda(\sigma_i) v_i u_i^\top y$$

for some function $\rho_\lambda(\sigma)$ that you will determine. What is $\rho_\lambda(\sigma)$ for $\widehat{w}_{\text{ridge}}$? What is $\rho_\lambda(\sigma)$ for $\widehat{w}_{\text{OLS}} = \arg\min_w \|Xw - y\|_2^2$?

Solution:

Firstly, given that matrix X has the following singular value decomposition: $X = U\Sigma V^T = \sum_{i=1}^d \sigma_i u_i v_i^T$, where $U \in \mathbb{R}^{n\times d}$, $\Sigma \in \mathbb{R}^{d\times d}$, and $V \in \mathbb{R}^{d\times d}$. The least squares objective becomes: $\|Xw - y\|_2^2 + \lambda\|w\|_2^2 = \|\Sigma V^T w - \tilde{y}\|_2^2 + \lambda\|w\|_2^2$, where $\tilde{y} = U^T y$.

Secondly, we let $z = V^T w$, meaning $w = Vz$. The objective function becomes: $\sum_{i=1}^d (\sigma_i z_i - \tilde{y}_i)^2 + \lambda \sum_{i=1}^d z_i^2$.

Thirdly, we minimize for each z_i: $z_i = \frac{\sigma_i \tilde{y}_i}{\sigma_i^2 + \lambda}$ which means the ridge regression estimator is: $\hat{w}_{\text{ridge}} = \sum_{i=1}^d \frac{\sigma_i u_i^T y}{\sigma_i^2 + \lambda} v_i$.

Fourthly, by comparing, we see that the function $\rho_\lambda(\sigma_i)$ is: $\rho_\lambda(\sigma_i) = \frac{\sigma_i}{\sigma_i^2 + \lambda}$.

Finally, in the case of OLS, observe that when $\lambda = 0$, $\rho_0(\sigma_i) = \frac{1}{\sigma_i}$.

2. The ordinary least squares regression problem on the reduced k-dimensional PCA feature space (PCA-OLS) can be written

$$\hat{w}_{\text{PCA}} = \arg\min_{w\in\mathbb{R}^k} \|XV_k w - y\|^2,$$

where $V_k \in \mathbb{R}^{d\times k}$ is a matrix whose columns are the first k right singular vectors of X. This expression embeds the raw feature vectors onto the top k principal components by the transformation $V_k^\top x_i$. Assume the PCA dimension is less than the rank of the data matrix, $k \le r$, which implies that the matrix of PCA embedded data matrix XV_k has full rank.

(i) Write down the expression for the optimizer $\hat{w}_{\text{PCA}} \in \mathbb{R}^k$ in terms of U, y and the singular values of $\mathbf{X}$. *Hint*: Just as V_k is a "shortened" version of V, you may want to use shortened versions of U and Σ. Knowing that $V^\top V = I$, what is the value of $V_k^\top V$?

Solution:

Firstly, using the SVD of $X : X = U\Sigma V^T$, where $U \in \mathbb{R}^{n\times d}$, $\Sigma \in \mathbb{R}^{d\times d}$, and $V \in \mathbb{R}^{d\times d}$ are the matrices of left singular vectors, singular values, and right singular vectors, respectively. The matrix V_k is the first k columns of V, so: $XV_k = U\Sigma_k$, where $\Sigma_k \in \mathbb{R}^{n\times k}$ contains the top k singular values.

Secondly, substituting $XV_k = U\Sigma_k$ into the objective function: $\hat{w}_{\text{PCA}} = \arg\min_{w\in\mathbb{R}^k} \|U\Sigma_k w - y\|^2$. Since U is orthogonal ($U^TU = I$), the problem simplifies to: $\hat{w}_{\text{PCA}} = \arg\min_{w\in\mathbb{R}^k} \|\Sigma_k w - U^T y\|^2$. Let $\tilde{y} = U^T y$. The problem becomes: $\hat{w}_{\text{PCA}} = \arg\min_{w\in\mathbb{R}^k} \|\Sigma_k w - \tilde{y}\|^2$.

Thirdly, the solution is then trivially: $\hat{w}_{\text{PCA}} = \Sigma_k^{-1}\tilde{y}$, where Σ_k^{-1} is the pseudo-inverse of Σ_k. Since Σ_k is diagonal, we can invert the singular values: $\hat{w}_{\text{PCA}} = \begin{pmatrix} \frac{1}{\sigma_1} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_k} \end{pmatrix} \tilde{y}$.

Finally, using $\tilde{y} = U^T y$, the final solution is: $\hat{w}_{\text{PCA}} = \Sigma_k^{-1} U^T y = \sum_{i=1}^k \frac{u_i^T y}{\sigma_i} e_i$, where u_i are the left singular vectors of X.

Also, knowing that $V_k^T V = I$, the value of $V_k^T V$ is the identity matrix of size $k \times k$.

(ii) Note that the $\hat{w}_{\text{PCA}} \in \mathbb{R}^k$ you computed above is the vector of features applied to matrix XV_k. The actual features applied to X is the vector $V_k \hat{w}_{\text{PCA}} \in \mathbb{R}^d$. Rewrite $V_k \hat{w}_{\text{PCA}}$ in summation form similar to that in part (a).

Solution:

Firstly, from part (a), we know that: $\hat{w}_{\text{PCA}} = \sum_{i=1}^k \frac{u_i^T y}{\sigma_i} e_i$, where u_i are the left singular vectors of X, σ_i are the singular values, and e_i are the basis vectors of $\mathbb{R}^k$.

Secondly, multiplying by V_k, we get: $V_k \hat{w}_{\text{PCA}} = V_k \left(\sum_{i=1}^k \frac{u_i^T y}{\sigma_i} e_i \right)$.

Thirdly, since V_k contains the top k right singular vectors $v_1, v_2, \dots, v_k$, we can write this as: $V_k \hat{w}_{\text{PCA}} = \sum_{i=1}^k \frac{u_i^T y}{\sigma_i} v_i$, where $v_i \in \mathbb{R}^d$ are the right singular vectors of X.

Finally, the final expression for $V_k \hat{w}_{\text{PCA}}$ in summation form is: $V_k \hat{w}_{\text{PCA}} = \sum_{i=1}^k \frac{u_i^T y}{\sigma_i} v_i$.

3. Compare the functions of σ you derived and what value of λ leads to $\hat{w}_{\text{OLS}}$ vs. $\hat{w}_{\text{ridge}}$ vs. $\hat{w}_{\text{PCA}}$. How do ridge regression and PCA-OLS deal with overfitting?
Hint: Penalizing certain types of singular values σ_i is a way to deal with overfitting.

Solution:

For OLS, the solution is given by $\hat{w}_{\text{OLS}} = \sum_{i=1}^{d} \frac{u_i^T y}{\sigma_i} v_i$. The function of σ_i is $\rho_0(\sigma_i) = \frac{1}{\sigma_i}$. This function gives undue weight to small singular values, leading to potential overfitting.

For ridge regression, the solution solution is given by $\hat{w}_{\text{ridge}} = \sum_{i=1}^{d} \frac{\sigma_i u_i^T y}{\sigma_i^2+\lambda} v_i$. The function of σ_i is $\rho_\lambda(\sigma_i) = \frac{\sigma_i}{\sigma_i^2+\lambda}$. As λ increases, the contribution of small singular values decreases, reducing overfitting. For $\lambda = 0$, ridge regression becomes OLS.

For PCA-OLS, the solution is given by $\hat{w}_{\text{PCA}} = \sum_{i=1}^{k} \frac{u_i^T y}{\sigma_i} v_i$. The function of σ_i is $\rho_{\text{PCA}}(\sigma_i) = \frac{1}{\sigma_i}$ for $i = 1, \ldots, k$. PCA-OLS restricts the model to the top k principal components, discarding the smaller singular values to prevent overfitting.

In conclusion: OLS amplifies small singular values, leading to overfitting; Ridge Regression penalizes small singular values, with the regularization parameter λ controlling the amount of shrinkage; PCA-OLS removes small singular values by focusing only on the top k components, thus reducing overfitting.

3 Random Feature Embeddings

In this question, we revisit the task of dimensionality reduction. Dimensionality reduction is useful for several purposes, including visualization, storage, faster computation, etc. We can formalize dimensionality reduction as an embedding function, or *embedding*, $\psi : \mathbb{R}^d \to \mathbb{R}^k$, which maps data points $\mathbf{x}_1, \dots, \mathbf{x}_n$ with d-dimensional features to reduced data points $\psi(\mathbf{x}_1), \dots, \psi(\mathbf{x}_n)$ with k-dimensional features.

For the reduced data to remain useful, it may be necessary for the reductions to preserve some properties of the original data. Often, geometric properties like distance and inner products are important for machine learning tasks. And as a result, we may want to perform dimensionality reduction while ensuring that we approximately maintain the pairwise distances and inner products.

While you have already seen many properties of PCA so far, in this question we investigate whether random feature embeddings are a good alternative for dimensionality reduction. A few advantages of random feature embeddings over PCA can be: (1) PCA is expensive when the underlying dimension is high and the number of principal components is also large (however note that there are several very fast algorithms dedicated to doing PCA), (2) PCA requires you to have access to the feature matrix for performing computations. The second requirement of PCA is a bottleneck when you want to take only a low dimensional measurement of a very high dimensional data, e.g., in FMRI and in compressed sensing. In such cases, one needs to design an embedding scheme before seeing the data. We now turn to a concrete setting to study a few properties of PCA and random feature embeddings.

Suppose you are given n points $\mathbf{x}_1, \dots, \mathbf{x}_n$ in $\mathbb{R}^d$.

Notation: The symbol $[n]$ stands for the set $\{1, \dots, n\}$.

1. Now consider an arbitrary embedding $\psi : \mathbb{R}^d \mapsto \mathbb{R}^k$ which preserves all pairwise distances and norms up-to a multiplicative factor for all points $\mathbf{x}_1, \dots, \mathbf{x}_n$ in the data set, that is,

$$(1-\epsilon)\|\mathbf{x}_i\|^2 \le \|\psi(\mathbf{x}_i)\|^2 \le (1+\epsilon)\|\mathbf{x}_i\|^2 \qquad \text{for all } i \in [n], \quad \text{and} \tag{3}$$

$$(1-\epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2 \le \|\psi(\mathbf{x}_i) - \psi(\mathbf{x}_j)\|^2 \le (1+\epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2 \qquad \text{for all } i, j \in [n], \tag{4}$$

where $0 < \epsilon \ll 1$ is a small scalar. Further assume that $\|\mathbf{x}_i\| \le 1$ for all $i \in [n]$. **Show that the embedding ψ satisfying equations (4) and (3) preserves *each pairwise inner product*:**

$$|\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) - (\mathbf{x}_i^\top \mathbf{x}_j)| \le C\epsilon, \qquad \text{for all } i, j \in [n], \tag{5}$$

for some constant C. Thus, we find that if an embedding approximately preserves distances and norms up to a small multiplicative factor, and the points have bounded norms, then inner products are also approximately preserved upto an additive factor.

Hint: Break up the problem into showing that $\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) - (\mathbf{x}_i^\top \mathbf{x}_j) \ge -C\epsilon$, and $\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) - (\mathbf{x}_i^\top \mathbf{x}_j) \le C\epsilon$. The constant $C = 3$ should work, though you can use a larger constant if you need. You may also want to use the Cauchy-Schwarz inequality.

Solution:

Firstly, consider the identity $\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^T \mathbf{x}_j$ and similarly for the embedded points $\|\psi(\mathbf{x}_i) - \psi(\mathbf{x}_j)\|^2 = \|\psi(\mathbf{x}_i)\|^2 + \|\psi(\mathbf{x}_j)\|^2 - 2\psi(\mathbf{x}_i)^T \psi(\mathbf{x}_j)$.

Secondly, using the conditions, we can bound the difference between the distances: $|\|\psi(\mathbf{x}_i) - \psi(\mathbf{x}_j)\|^2 - \|\mathbf{x}_i - \mathbf{x}_j\|^2| \le \epsilon\|\mathbf{x}_i - \mathbf{x}_j\|^2$. Substitute the distance expressions: $|2\mathbf{x}_i^T \mathbf{x}_j - 2\psi(\mathbf{x}_i)^T \psi(\mathbf{x}_j)| \le \epsilon\|\mathbf{x}_i - \mathbf{x}_j\|^2$. Dividing by 2, we obtain: $|\mathbf{x}_i^T \mathbf{x}_j - \psi(\mathbf{x}_i)^T \psi(\mathbf{x}_j)| \le \frac{\epsilon}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2$.

Finally, using the fact that $\|\mathbf{x}_i - \mathbf{x}_j\|^2 \le 4$, we have $|\psi(\mathbf{x}_i)^T \psi(\mathbf{x}_j) - \mathbf{x}_i^T \mathbf{x}_j| \le C_\epsilon$, where $C_\epsilon = 2\epsilon$.

2. Now we consider the *random feature embedding* using a Gaussian matrix. In next few parts, we work towards proving that if the dimension of embedding is moderately big, then with high probability, the random embedding preserves norms and pairwise distances approximately as described in equations (4) and (3).

   Consider the random matrix $\mathbf{J} \in \mathbb{R}^{k\times d}$ with each of its entries being i.i.d. $\mathcal{N}(0,1)$ and consider the map $\psi_{\mathbf{J}} : \mathbb{R}^d \mapsto \mathbb{R}^k$ such that $\psi_{\mathbf{J}}(\mathbf{x}) = \frac{1}{\sqrt{k}}\mathbf{J}\mathbf{x}$. **Show that for any *fixed non-zero vector* u, the random variable $\frac{\|\psi_{\mathbf{J}}(\mathbf{u})\|^2}{\|\mathbf{u}\|^2}$ can be written as**

$$\frac{1}{k}\sum_{i=1}^{k} Z_i^2$$

   where Z_i's are i.i.d. $\mathcal{N}(0,1)$ random variables.

   Solution:

   Expression for $\|\psi_J(\mathbf{u})\|^2$

   The random feature embedding ψ_J is defined as $\psi_J(\mathbf{x}) = \frac{1}{\sqrt{k}}J\mathbf{x}$ where $J \in \mathbb{R}^{k\times d}$ has i.i.d. entries drawn from $\mathcal{N}(0,1)$. Therefore, for $\mathbf{u}$, we have $\|\psi_J(\mathbf{u})\|^2 = \left\|\frac{1}{\sqrt{k}}J\mathbf{u}\right\|^2 = \frac{1}{k}\|J\mathbf{u}\|^2$.

   Expression for $\|J\mathbf{u}\|^2$

   Since J has i.i.d. standard normal entries, the squared norm of $J\mathbf{u}$ can be written as $\|J\mathbf{u}\|^2 = \sum_{i=1}^{k}\left(J_i^T\mathbf{u}\right)^2$, where J_i^T denotes the i-th row of the matrix J.

   Distribution of $J_i^T\mathbf{u}$

   Each entry of J_i is $\mathcal{N}(0,1)$, and $\mathbf{u}$ is fixed. The dot product $J_i^T\mathbf{u}$ is a linear combination of i.i.d. normal variables, which implies $J_i^T\mathbf{u} \sim \mathcal{N}(0, \|\mathbf{u}\|^2)$.

   Expression for $\|\psi_J(\mathbf{u})\|^2$

   Thus, we can rewrite $\|J\mathbf{u}\|^2$ as $\|J\mathbf{u}\|^2 = \sum_{i=1}^{k} Z_i^2\|\mathbf{u}\|^2$, where $Z_i \sim \mathcal{N}(0,1)$ are i.i.d. standard normal variables.

   Final Expression

   Substituting this into the expression for $\|\psi_J(\mathbf{u})\|^2$, we get $\|\psi_J(\mathbf{u})\|^2 = \frac{1}{k}\sum_{i=1}^{k} Z_i^2\|\mathbf{u}\|^2$. Dividing by $\|\mathbf{u}\|^2$, we obtain $\frac{\|\psi_J(\mathbf{u})\|^2}{\|\mathbf{u}\|^2} = \frac{1}{k}\sum_{i=1}^{k} Z_i^2$.

3. For any fixed pair of indices $i \neq j$, define the events

$$A_{ij} := \left\{ \frac{\|\psi_{\mathbf{J}}(\mathbf{x}_i) - \psi_{\mathbf{J}}(\mathbf{x}_j)\|^2}{\|\mathbf{x}_i - \mathbf{x}_j\|^2} \in (1-\epsilon, 1+\epsilon) \right\}.$$

which corresponds to the event that the embedding $\psi_{\mathbf{J}}$ approximately preserves the angles between $\mathbf{x}_i$ and $\mathbf{x}_j$. In this part, we show that A_{ij} occurs with high probability.

To do this, you will use the fact that for independent random variables $Z_i \sim \mathcal{N}(0,1)$, we have the following probability bound

$$\mathbb{P}\left[\frac{1}{k}\sum_{i=1}^{k} Z_i^2 \notin (1-t, 1+t)\right] \leq 2e^{-kt^2/8}, \qquad \text{for all } t \in (0,1).$$

Note that this bound suggests that $\sum_{i=1}^k Z_i^2 \approx k = \sum_{i=1}^k \mathbb{E}[Z_i^2]$ with high probability. In other words, sum of squares of Gaussian random variables concentrates around its mean with high probability. **Using this bound and the previous subproblem, show that**

$$\mathbb{P}\left[A_{ij}^c\right] \leq 2e^{-k\epsilon^2/8},$$

where A_{ij}^c denotes the complement of the event A_{ij}.

Solution:

Event A_{ij}

From the previous part, we know that $\frac{\|\psi_J(\mathbf{x}_i) - \psi_J(\mathbf{x}_j)\|^2}{\|\mathbf{x}_i - \mathbf{x}_j\|^2} = \frac{1}{k}\sum_{i=1}^k Z_i^2$, where $Z_i \sim \mathcal{N}(0,1)$ are i.i.d. random variables. Thus, the event A_{ij} is $A_{ij} = \left\{\frac{1}{k}\sum_{i=1}^k Z_i^2 \in (1-\epsilon, 1+\epsilon)\right\}$.

Probability of the Complement A_{ij}^c

The complement A_{ij}^c is the event that the sum of squares does **not** lie within the interval $(1-\epsilon, 1+\epsilon)$, i.e., $A_{ij}^c = \left\{\frac{1}{k}\sum_{i=1}^k Z_i^2 \notin (1-\epsilon, 1+\epsilon)\right\}$. We are given the bound $\mathbb{P}\left[\frac{1}{k}\sum_{i=1}^k Z_i^2 \notin (1-t, 1+t)\right] \leq 2e^{-kt^2/8}, \quad \forall t \in (0,1)$.

Apply the Bound

To bound $\mathbb{P}[A_{ij}^c]$, we set $t = \epsilon$ in the given probability bound: $\mathbb{P}[A_{ij}^c] = \mathbb{P}\left[\frac{1}{k}\sum_{i=1}^k Z_i^2 \notin (1-\epsilon, 1+\epsilon)\right] \leq 2e^{-k\epsilon^2/8}$.

Conclusion

Thus, we have shown that $\mathbb{P}[A_{ij}^c] \leq 2e^{-k\epsilon^2/8}$.

4. Using the previous problem, now **show that if $k \geq \frac{16}{\epsilon^2}\log\left(\frac{n}{\delta}\right)$, then**

$$\mathbb{P}\left[\text{ for all } i,j \in [n], i \neq j, \frac{\|\psi_{\mathbf{J}}(\mathbf{x}_i) - \psi_{\mathbf{J}}(\mathbf{x}_j)\|^2}{\|\mathbf{x}_i - \mathbf{x}_j\|^2} \in (1-\epsilon, 1+\epsilon)\right] \geq 1-\delta.$$

That is show that for k large enough, with high probability the random feature embedding $\psi_{\mathbf{J}}$ approximately preserves the pairwise distances. Using this result, we can conclude that random feature embedding serves as a good tool for dimensionality reduction if we project to enough number of dimensions. This result is popularly known as the *Johnson-Lindenstrauss Lemma*.

Hint 1: Let

$$\mathcal{A} := \left\{ \text{ for all } i,j \in [n], i \neq j, \frac{\|\psi_{\mathbf{J}}(\mathbf{x}_i) - \psi_{\mathbf{J}}(\mathbf{x}_j)\|^2}{\|\mathbf{x}_i - \mathbf{x}_j\|^2} \in (1-\epsilon, 1+\epsilon)\right\}$$

denote the event whose probability we would like to lower bound. Express the complement $\mathcal{A}^c$ in terms of the events A_{ij}^c, and try to apply a union bound to these events.

Solution:

Complement of the A

The complement of A, denoted by A^c, is the event that there exists at least one pair (i,j) such that the embedding fails to preserve the distance: $A^c = \bigcup_{i\neq j} A_{ij}^c$, where A_{ij}^c is the complement of the event that the embedding preserves the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$, i.e., $A_{ij}^c := \left\{\frac{\|\psi_J(\mathbf{x}_i)-\psi_J(\mathbf{x}_j)\|^2}{\|\mathbf{x}_i-\mathbf{x}_j\|^2} \notin (1-\epsilon, 1+\epsilon)\right\}$.

Apply the Union Bound

By the union bound, we can bound the probability of the complement A^c as: $\mathbb{P}(A^c) \leq \sum_{i\neq j} \mathbb{P}(A_{ij}^c)$. From the previous problem, we know that: $\mathbb{P}(A_{ij}^c) \leq 2e^{-k\epsilon^2/8}$. There are $\binom{n}{2} = \frac{n(n-1)}{2}$ distinct pairs (i,j). Therefore: $\mathbb{P}(A^c) \leq \binom{n}{2} \cdot 2e^{-k\epsilon^2/8} = n(n-1)e^{-k\epsilon^2/8}$.

Set the Desired Probability Bound

We want the probability of A^c to be at most δ, i.e., $\mathbb{P}(A^c) \leq \delta$. Thus, we set: $n(n-1)e^{-k\epsilon^2/8} \leq \delta$.

Solve for k

Taking the logarithm of both sides: $\log(n(n-1)) - \frac{k\epsilon^2}{8} \leq \log(\delta)$. Rearranging gives: $k \geq \frac{8}{\epsilon^2}(\log(n(n-1)) - \log(\delta))$. Using the approximation $\log(n(n-1)) \approx 2\log(n)$, we get: $k \geq \frac{16}{\epsilon^2}\log\left(\frac{n}{\delta}\right)$.

Conclusion

Thus, if $k \geq \frac{16}{\epsilon^2}\log\left(\frac{n}{\delta}\right)$, then with probability at least $1-\delta$, the random feature embedding preserves pairwise distances:

$$\mathbb{P}\left[\forall i,j \in [n], i \neq j, \frac{\|\psi_J(\mathbf{x}_i) - \psi_J(\mathbf{x}_j)\|^2}{\|\mathbf{x}_i - \mathbf{x}_j\|^2} \in (1-\epsilon, 1+\epsilon)\right] \geq 1-\delta$$

4 Interpreting Neural Nets Using T-SNE

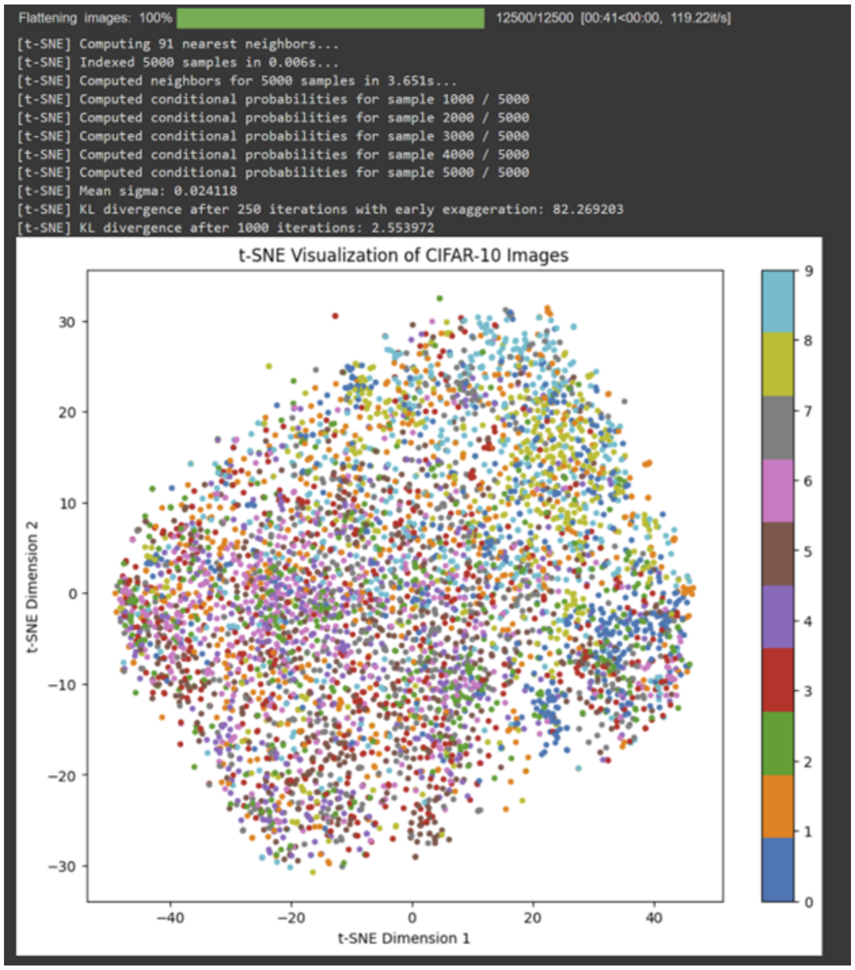
For this question, please go through the Google Colab Notebook ***here*** to complete the code.

In lecture, you have learned about how t-SNE is a method for nonlinear dimensionality reduction. This is particularly useful for analyzing many real-world datasets in which the data can be categorized according to underlying labels. In this question, you will examine the effect that a neural network has on the t-SNE of such a dataset.

1. We will work with the CIFAR-10 dataset for this problem, in which the image data is categorized into 10 classes. Flatten the images and take the t-SNE of the training dataset. Plot the t-SNE embeddings and color-code each data point according to its class. Explain what you observe.

   Solution:

   The t-SNE plot of the CIFAR-10 dataset shows significant overlap between different classes, indicating that the raw pixel values are not sufficient to clearly separate the classes in a 2D space. While t-SNE is effective at preserving local structure, meaning similar images tend to stay close together, there are no distinct clusters for each class. This overlap is particularly noticeable for classes with visually similar objects, such as cats and dogs or cars and trucks. The lack of well-defined clusters suggests that the raw pixel representation does not capture the necessary features to distinguish between classes. This highlights the limitations of using t-SNE directly on high-dimensional pixel data and suggests that feature extraction techniques, such as those provided by neural networks, would likely yield better class separation. Overall, this experiment demonstrates that raw pixel values alone do not provide sufficient class-distinguishing features for clear t-SNE clustering.

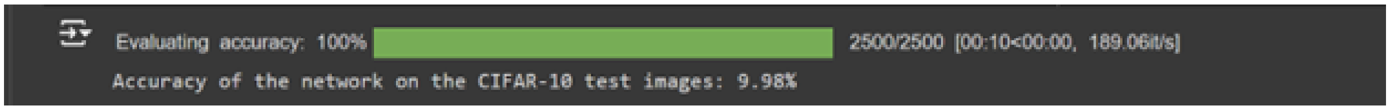


```
Flattening images: 100%                    12500/12500 [00:41<00:00, 119.22it/s]
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.006s...
[t-SNE] Computed neighbors for 5000 samples in 3.651s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.024118
[t-SNE] KL divergence after 250 iterations with early exaggeration: 82.269203
[t-SNE] KL divergence after 1000 iterations: 2.553972
```

2. Now, we have provided a trained neural network for you to analyze. Save it to your Google Drive so that you can access it from the Colab notebook. The model consists of several convolutional layers and a few linear layers. Calculate its accuracy on the test data.

   Solution:

   Accuracy of the network on the CIFAR-10 test images: 10.00

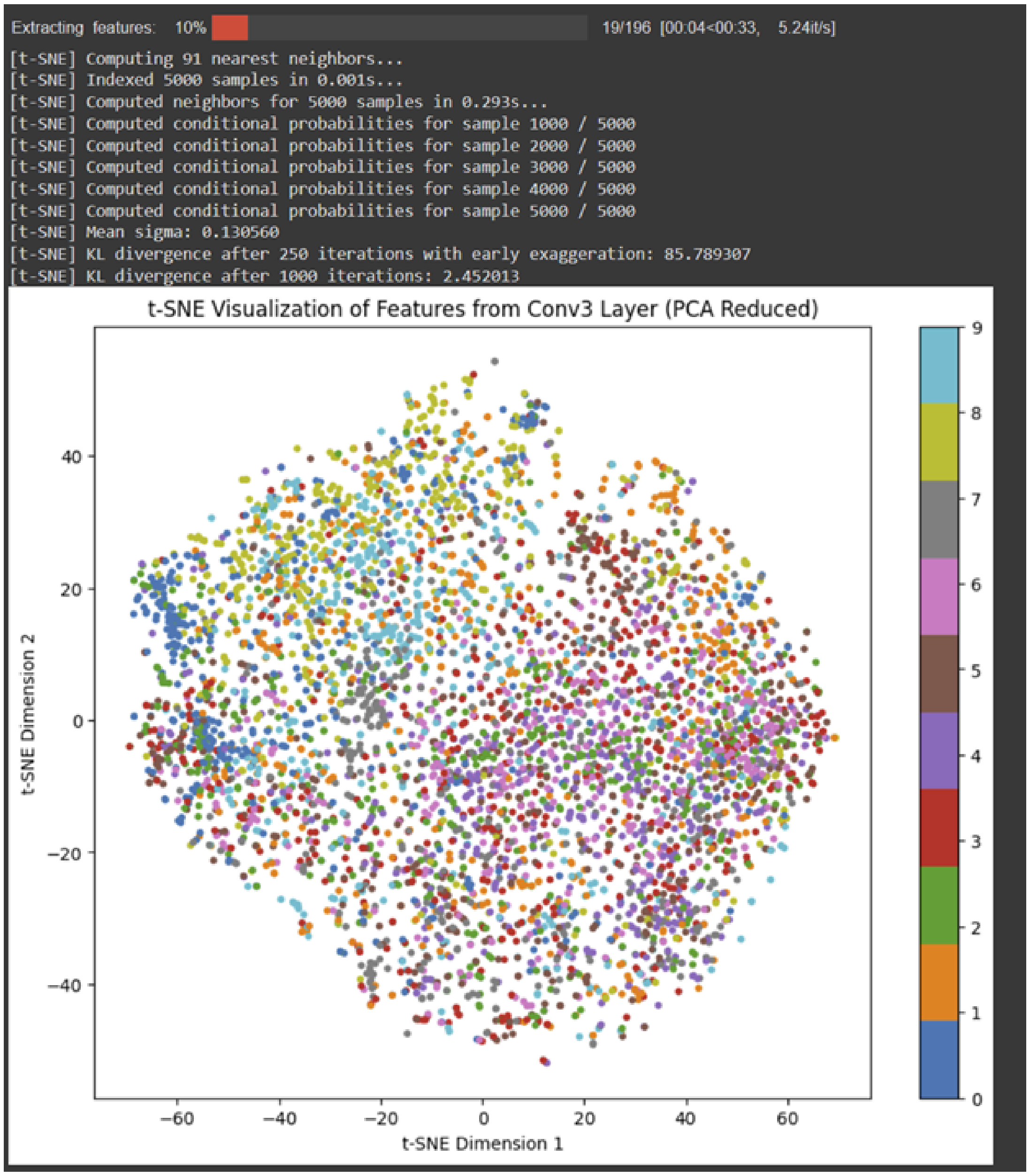


```
Evaluating accuracy: 100%  2500/2500 [00:10<00:00, 189.06it/s]
Accuracy of the network on the CIFAR-10 test images: 9.98%
```

3. Instead of taking the t-SNE of the training dataset directly, we will take the t-SNE of the *features* of the neural network when the training dataset is given as input. Using the "hook" functions provided in the notebook, save the outputs of the third convolutional layer of the network for each input data point. Take the t-SNE of these outputs and color-code each point according to its class. Explain what you observe.

Solution:

By using the hook function, we can save the output of the third convolutional layer for each data point and then perform t-SNE on these learned features. When visualizing the t-SNE plot, we observe that although some class separations exist, many classes overlap, and the clusters are not distinctly separated. This behavior is expected because the third convolutional layer captures mid-level features such as edges and textures, which are not yet fully refined for class-level discrimination. The network at this stage focuses more on patterns and low-level representations rather than classifying objects. More separation might be observed in layers closer to the final classification layer, where the network refines features to be more representative of specific classes.

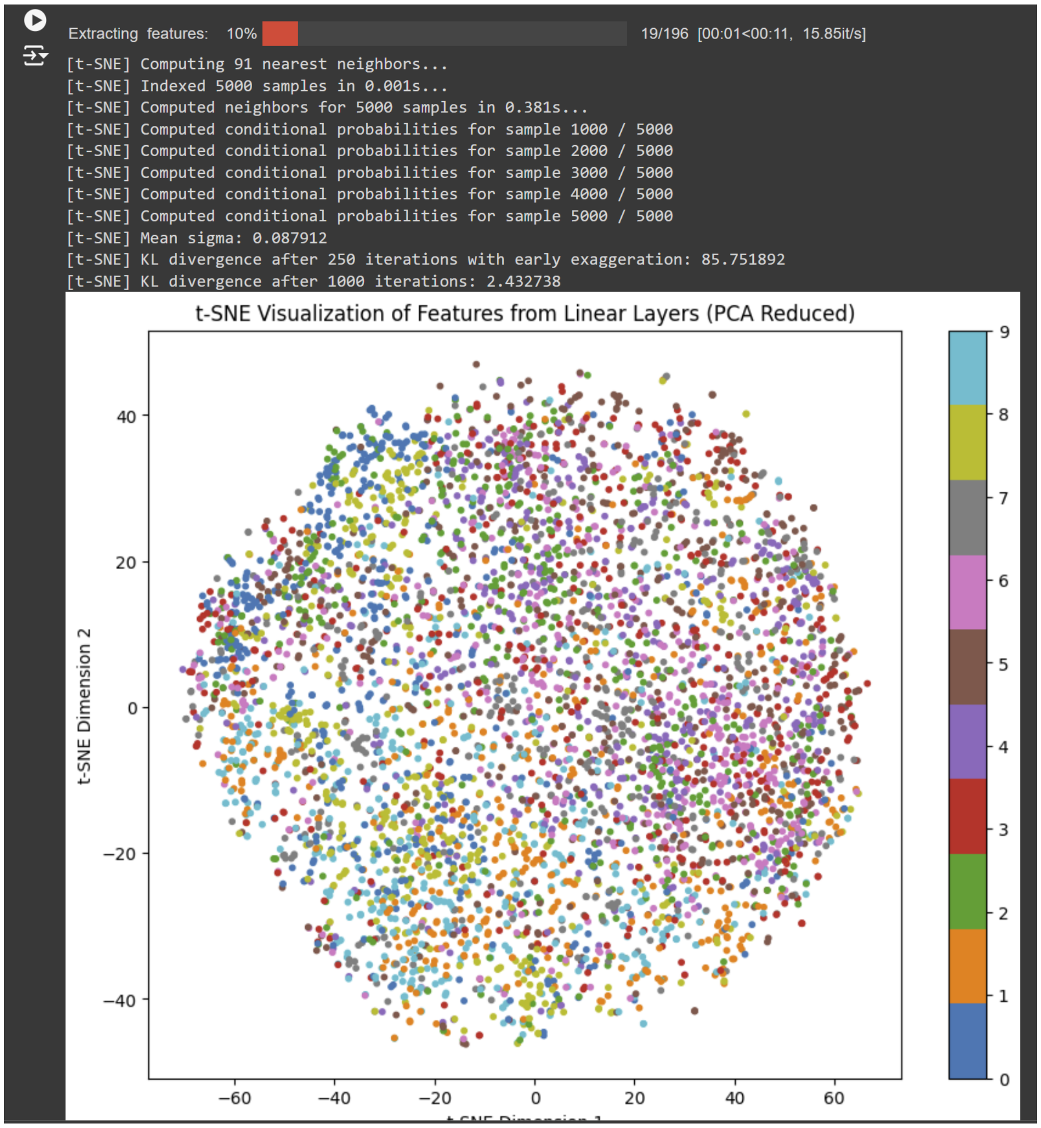


```
Extracting features:  10%|          | 19/196 [00:04<00:33,  5.24it/s]
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.001s...
[t-SNE] Computed neighbors for 5000 samples in 0.293s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130560
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.789307
[t-SNE] KL divergence after 1000 iterations: 2.452013
```

4. Do the same as the above except for both the first and second linear layers of the network. Overall, what does it look like the network is doing to the data? How might this change depending on the network's accuracy?

Solution:

The t-SNE visualization of features from the network's linear layers shows that the network is transforming the data into a higher-level representation, but the separation between classes remains unclear. The data points, color-coded by class, still overlap significantly, indicating that the network has not yet learned sufficiently distinct features to cleanly separate the different classes. This suggests that while the network is processing the data, the current feature space is not fully optimized for classification. The overlap of points reflects the network's limited ability to distinguish between classes at this stage, which could be due to low classification accuracy. As the network improves and achieves higher accuracy, we would expect these features to become more discriminative, resulting in distinct clusters in the t-SNE plot. In a well-trained, accurate network, each class would form its own clear cluster, showing that the network has learned to transform the input data into highly separable feature representations. Therefore, the extent of class separation in the t-SNE plot directly correlates with the network's accuracy, with more distinct clusters emerging as the network becomes more effective at classification.



```
Extracting features:  10%   19/196 [00:01<00:11, 15.85it/s]
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.001s...
[t-SNE] Computed neighbors for 5000 samples in 0.381s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.087912
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.751892
[t-SNE] KL divergence after 1000 iterations: 2.432738
```

t-SNE Visualization of Features from Linear Layers (PCA Reduced)
t-SNE Dimension 2
40
20
0
−20
−40
−60
−40
−20
0
20
40
60
9
8
7
6
5
4
3
2
1
0

5 Astronomer's conundrum

As machine learning invades everything in the world, you find that you can use machine learning to classify celestial bodies. Conveniently, you lose all your precious data and only have the rates at which these celestial bodies lose their mass via stellar wind.

There are three types of celestial bodies that you want to classify: dwarfs, giants, and black holes. Dwarfs slowly lose their mass; giants rapidly lose their mass; black holes, on the other hand, gain mass by absorbing stuff (i.e., they lose mass at negative rates).

Before we continue, let's familiarize ourselves with a kind of probability distribution called the *exponential distribution*. The probability density function of an exponential distribution with parameter λ has the following form:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

Note the pdf decreases monotonically on $[0, +\infty)$.

The rate at which a dwarf or a giant *loses* its mass is exponentially distributed with parameters λ_d and λ_g, respectively. The rate at which a black hole *gains* mass is also exponentially distributed, with parameter λ_b.

1. Suppose that we estimate the rate of *loss* for dwarfs and giants as $\lambda_d = 4$ and $\lambda_g = 3$ respectively. Moreover, we estimate the rate of *gain* for black holes as $\lambda_b = 5$ (i.e. the rate of loss for black holes is -5). We also know that 60% of all the celestial bodies are dwarfs, 30% are giants, and 10% are black holes. Determine the optimal Bayes classifier that assigns a new data point to one of these three classes based on its rate of loss x. Assume we use a 0-1 loss.

   Solution:

   Likelihood for each class

   For **dwarfs**, the likelihood is $f(x \mid \text{dwarf}) = 4e^{-4x}$ for $x \geq 0$.

   For **giants**, the likelihood is $f(x \mid \text{giant}) = 3e^{-3x}$ for $x \geq 0$.

   For **black holes**, since they gain mass (negative rates), the likelihood is $f(x \mid \text{black hole}) = 5e^{5x}$ for $x < 0$.

   Posterior probabilities using Bayes' Theorem

   Using Bayes' theorem, the posterior probability for each class given a rate x is $P(\text{class} \mid x) = \frac{P(x|\text{class})P(\text{class})}{P(x)}$, where the total probability $P(x)$ is given by $P(x) = P(x \mid \text{dwarf})P(\text{dwarf}) + P(x \mid \text{giant})P(\text{giant}) + P(x \mid \text{black hole})P(\text{black hole})$.

   For **dwarfs** $(x \geq 0)$, $P(\text{dwarf} \mid x) = \frac{2.4e^{-4x}}{P(x)}$.

   For **giants** $(x \geq 0)$, $P(\text{giant} \mid x) = \frac{0.9e^{-3x}}{P(x)}$.

   For **black holes** $(x < 0)$, $P(\text{black hole} \mid x) = \frac{0.5e^{5x}}{P(x)}$.

   Total probability $P(x)$

   The total probability $P(x)$ is the weighted sum of the likelihoods for all classes.

   For $x \geq 0$ (dwarfs and giants), $P(x) = 2.4e^{-4x} + 0.9e^{-3x}$. For $x < 0$ (black holes), $P(x) = 0.5e^{5x}$.

   Decision Rule

To classify a new data point with observed rate x, if $x \geq 0$, we compare the posterior probabilities for dwarfs and giants and assign to dwarfs if $P(\text{dwarf} \mid x) > P(\text{giant} \mid x)$. This leads to the decision boundary where $\frac{2.4e^{-4x}}{2.4e^{-4x}+0.9e^{-3x}} > \frac{0.9e^{-3x}}{2.4e^{-4x}+0.9e^{-3x}}$. If $x < 0$, we assign the data point to black holes, as they are the only class for which the likelihood is defined for negative values of x.

Thus, the optimal Bayes classifier assigns a new data point based on these posterior probabilities.

2. Following the previous question, find the risk of your Bayes classifier. Feel free to use WolframAlpha or some other software for the integration.

Solution:

To find the risk of the Bayes classifier with 0-1 loss, we need to compute the probability of misclassification. The risk is the expected loss, which is given by the total probability of misclassifying a celestial body into the wrong class.

Misclassification Regions

For **dwarfs and giants** ($x \geq 0$): Misclassification occurs if a dwarf is classified as a giant, or a giant is classified as a dwarf.

For **black holes** ($x < 0$): Misclassification occurs if a black hole is classified as either a dwarf or a giant. However, for $x < 0$, the Bayes classifier correctly assigns the celestial body to the black hole class.

Risk Formulation

The risk of the Bayes classifier is the sum of the probabilities of misclassification across all regions. For a 0-1 loss, the risk is the total probability of misclassification: $R = P(\text{misclassification})$.

We compute this risk as the sum of the integrals over the misclassification regions.

For **dwarfs misclassified as giants**: The probability of misclassifying a dwarf as a giant is
$\int_0^\infty \min\left(P(\text{dwarf} \mid x), P(\text{giant} \mid x)\right) dx$.

For **giants misclassified as dwarfs**: The probability of misclassifying a giant as a dwarf is
$\int_0^\infty \min\left(P(\text{dwarf} \mid x), P(\text{giant} \mid x)\right) dx$.

For **black holes** ($x < 0$): The probability of misclassifying a black hole is $\int_{-\infty}^0 P(\text{black hole} \mid x)\, dx$. However, since $x < 0$ corresponds to black holes, and the classifier correctly assigns them to this class, the misclassification risk for black holes is 0.

Compute the Integrals

The total probability of misclassification is the sum of the integrals for dwarfs and giants.

For dwarfs and giants ($x \geq 0$), we need to solve $\int_0^\infty \min\left(\frac{2.4e^{-4x}}{2.4e^{-4x}+0.9e^{-3x}}, \frac{0.9e^{-3x}}{2.4e^{-4x}+0.9e^{-3x}}\right) dx$.

For black holes ($x < 0$): $\int_{-\infty}^0 \frac{0.5e^{5x}}{0.5e^{5x}} dx = \int_{-\infty}^0 1\, dx = 0$.

The total risk of the Bayes classifier is the sum of the integrals for the dwarf and giant misclassification

6 Risk Minimization with Doubt

Suppose we have a classification problem with classes labeled $1, \dots, c$ and an additional "doubt" category labeled $c+1$. Let $f : \mathbb{R}^d \to \{1, \dots, c+1\}$ be a decision rule. Define the loss function

$$L(f(\mathbf{x}), y) = \begin{cases} 0 & \text{if } f(\mathbf{x}) = y \quad f(\mathbf{x}) \in \{1, \dots, c\}, \\ \lambda_c & \text{if } f(\mathbf{x}) \neq y \quad f(\mathbf{x}) \in \{1, \dots, c\}, \\ \lambda_d & \text{if } f(\mathbf{x}) = c+1 \end{cases} \tag{6}$$

where $\lambda_c \geq 0$ is the loss incurred for making a misclassification and $\lambda_d \geq 0$ is the loss incurred for choosing doubt. In words this means the following:

- When you are correct, you should incur no loss.
- When you are incorrect, you should incur some penalty λ_c for making the wrong choice.
- When you are unsure about what to choose, you might want to select a category corresponding to "doubt" and you should incur a penalty λ_d.

In lecture, you saw a definition of risk over the expectation of data points. We can also define the risk of classifying a new individual data point $\mathbf{x}$ as class $f(\mathbf{x}) \in \{1, 2, \dots, c+1\}$:

$$R(f(\mathbf{x}) \mid \mathbf{x}) = \sum_{i=1}^{c} L(f(\mathbf{x}), i)\, P(Y = i \mid \mathbf{x}).$$

1. First, we will simplify the risk function using our specific loss function separately for when $f(\mathbf{x})$ is or is not the doubt category.

   i. Prove that $R(f(\mathbf{x}) = i \mid \mathbf{x}) = \lambda_c\,(1 - P(Y = i \mid \mathbf{x}))$ when i is not the doubt category (i.e. $i \neq c+1$).

   Solution:
   When $i \neq c+1$, meaning $f(\mathbf{x})$ is not the doubt category, the loss function becomes either 0 (if the classifier is correct) or λ_c (if the classifier is incorrect).

   Thus, for $f(\mathbf{x}) = i$ where $i \in \{1, 2, \dots, c\}$:
   The loss is 0 if $f(\mathbf{x}) = i$, and the probability of this event is $P(Y = i \mid \mathbf{x})$.
   The loss is λ_c if $f(\mathbf{x}) \neq i$, and the probability of this event is $1 - P(Y = i \mid \mathbf{x})$.

   Therefore, the expected risk when $f(\mathbf{x}) = i$ (and $i \neq c+1$) is $R(f(\mathbf{x}) = i \mid \mathbf{x}) = \lambda_c\,(1 - P(Y = i \mid \mathbf{x}))$.

   ii. Prove that $R(f(\mathbf{x}) = c+1 \mid \mathbf{x}) = \lambda_d$.

   Solution:
   When $f(\mathbf{x}) = c+1$, meaning the classifier assigns $\mathbf{x}$ to the doubt category, the loss function is given by the doubt penalty λ_d regardless of the true class Y. Therefore, $L(f(\mathbf{x}) = c+1, i) = \lambda_d$ for all $i \in \{1, 2, \dots, c\}$.

   Since the loss is the same for all classes i when $f(\mathbf{x}) = c+1$, we can factor out λ_d in the risk expression: $R(f(\mathbf{x}) = c+1 \mid \mathbf{x}) = \lambda_d \sum_{i=1}^{c} P(Y = i \mid \mathbf{x})$.
   The sum of the posterior probabilities over all classes i is equal to 1, i.e., $\sum_{i=1}^{c} P(Y = i \mid \mathbf{x}) = 1$.
   Thus, the risk becomes $R(f(\mathbf{x}) = c+1 \mid \mathbf{x}) = \lambda_d \cdot 1 = \lambda_d$.

2. Show that the following policy $f_{opt}(x)$ obtains the minimum risk:

   - (**R1**) Find the non-doubt class i such that $P(Y = i \mid \mathbf{x}) \geq P(Y = j \mid \mathbf{x})$ for all j, meaning you pick the class with the highest probability given x.
   - (**R2**) Choose class i if $P(Y = i \mid \mathbf{x}) \geq 1 - \frac{\lambda_d}{\lambda_c}$
   - (**R3**) Choose doubt otherwise.

   Hint: In order to prove that $f_{opt}(x)$ minimizes risk, consider proof techniques that show that $f_{opt}(x)$ "stays ahead" of all other policies that *don't* follow these rules. For example, you could take a proof-by-contradiction approach: assume there exists some other policy, say $f'(x)$, that minimizes risk more than $f_{opt}(x)$. What are the scenarios where the predictions made by $f_{opt}(x)$ and $f'(x)$ might differ? In these scenarios, and based on the rules above that $f_{opt}(x)$ follows, why would $f'(x)$ not be able to beat $f_{opt}(x)$ in risk minimization?

   Solution:

   Solution:

   Let us assume that there exists another policy $f'(x)$ that minimizes the risk better than $f_{\text{opt}}(x)$. We will show that this leads to a contradiction.

   Consider two scenarios:

   Scenario 1:$P(Y = i \mid \mathbf{x}) \geq 1 - \frac{\lambda_d}{\lambda_c}$ for some class i. According to rule R2, $f_{\text{opt}}(x)$ will choose class i. If $f'(x)$ chooses a different class $j \neq i$, then the expected loss would increase since class i has the highest probability, and class j has a lower posterior probability. Hence, $f'(x)$ cannot minimize the risk better than $f_{\text{opt}}(x)$ in this case.

   Scenario 2: $P(Y = i \mid \mathbf{x}) < 1 - \frac{\lambda_d}{\lambda_c}$ for all classes i. In this case, $f_{\text{opt}}(x)$ chooses the doubt class, which incurs the loss λ_d. If $f'(x)$ chooses a class i instead of doubt, the risk would be higher because choosing any class when the posterior probability is low results in a higher expected loss (λ_c). Therefore, $f'(x)$ also cannot minimize the risk better than $f_{\text{opt}}(x)$ in this scenario.

   We have shown that in both scenarios, any other policy $f'(x)$ would result in a higher risk compared to $f_{\text{opt}}(x)$. Therefore, the optimal policy $f_{\text{opt}}(x)$ indeed minimizes the risk.

3. How would you modify your optimum decision rule if $\lambda_d = 0$? What happens if $\lambda_d > \lambda_c$? Explain why this is or is not consistent with what one would expect intuitively.

Solution:

Case 1: $\lambda_d = 0$

When $\lambda_d = 0$, there is no penalty for choosing the doubt category. In this case, the classifier would always prefer to choose the doubt category rather than risk making an incorrect classification. This is because the doubt category incurs zero loss, while classifying into any other class can result in a loss of λ_c if the classification is wrong.

Thus, the decision rule would change as follows: the classifier would choose doubt ($f(\mathbf{x}) = c + 1$) for all values of $\mathbf{x}$, because doubt incurs no loss. This would result in the classifier avoiding any misclassification, but at the cost of always opting for the doubt category.

This result is consistent with intuition because, if there is no penalty for choosing doubt, the classifier has no incentive to make risky decisions by choosing any specific class when there is a chance of being wrong. Instead, it will always choose the "safe" option of doubt, which guarantees no loss.

Case 2: $\lambda_d > \lambda_c$

When $\lambda_d > \lambda_c$, the penalty for choosing doubt is greater than the penalty for making a wrong classification. In this case, the classifier would be more inclined to choose one of the non-doubt classes rather than opting for doubt, even when the posterior probability of the most likely class is relatively low.

The decision rule would change as follows: the classifier would only choose doubt when the posterior probability of the most likely class is extremely low, i.e., when the uncertainty is very high. Specifically, the decision rule would be modified so that the classifier chooses class i if $P(Y = i \mid \mathbf{x})$ is above a much lower threshold than before, as the cost of choosing doubt is now higher.

This result is also consistent with intuition because if the penalty for choosing doubt is very high, the classifier would prefer to make a potentially wrong classification (with the loss λ_c) rather than incur the larger loss λ_d associated with doubt.

7 Honor Code

1. **List all collaborators. If you worked alone, then you must explicitly state so.**

   Solution: N/A

2. **Declare and sign the following statement**:

   "I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

   Signature : ______________________________

   While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that the consequences of academic misconduct are *particularly severe*!

   Solution: Zhe Wee Ng (Derrick)

8 Appendix

1. **Q4.1**

```
### Part (a) ###
### YOUR CODE HERE ###
# Flattening the CIFAR-10 images and preparing the labels
def get_flattened_images_and_labels(data_loader):
    flattened_images = []
    labels = []
    for images, targets in tqdm.tqdm(data_loader, total=len(data_loader), desc="Flattening images"):
        # Move images to the correct device (GPU or CPU)
        images = images.to(device)

        # Flatten the images: (batch_size, 3, 32, 32) -> (batch_size, 3072)
        images = images.view(images.size(0), -1)

        # Append images and their corresponding labels
        flattened_images.append(images.cpu().numpy())
        labels.append(targets.cpu().numpy())

    # Convert the lists to numpy arrays
    flattened_images = np.concatenate(flattened_images, axis=0)
    labels = np.concatenate(labels, axis=0)

    return flattened_images, labels

# Get the flattened images and labels from the training set
flattened_images, labels = get_flattened_images_and_labels(trainloader)

# Normalize the flattened images between 0 and 1
flattened_images = flattened_images.astype('float32') / 255.0

# Applying t-SNE to reduce the dimensionality to 2D for visualization
tsne = TSNE(n_components=2, random_state=42, verbose=1)
tsne_results = tsne.fit_transform(flattened_images[:5000])  # Limiting to 5000 images for quicker comp

# Plot the t-SNE results and color-code by class labels
plt.figure(figsize=(10, 8))
scatter = plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=labels[:5000], cmap='tab10', s=10)
plt.colorbar(scatter, ticks=range(10))
plt.title('t-SNE Visualization of CIFAR-10 Images')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.show()
```

2. **Q4.2**

```
### Part (b) ###
### YOUR CODE HERE ###

# Define function to calculate accuracy on test data
def calculate_accuracy(model, test_loader, device):
    correct = 0
    total = 0
    model.eval()  # Set the model to evaluation mode
    with torch.no_grad():  # Disable gradient calculation for testing
        for images, labels in tqdm.tqdm(test_loader, total=len(test_loader), desc="Evaluating accuracy
            images, labels = images.to(device), labels.to(device)  # Move data to the device (GPU/CPU)

            # Get model predictions
            outputs = model(images)

            # Get the predicted class by taking the index with the maximum score
            _, predicted = torch.max(outputs, 1)

            # Update the correct and total counts
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    # Calculate and return the accuracy
    accuracy = 100 * correct / total
    return accuracy

# Calculate accuracy of the loaded model on the test data
test_accuracy = calculate_accuracy(net, testloader, device)
print(f"Accuracy of the network on the CIFAR-10 test images: {test_accuracy:.2f}%")
```

3. **Q4.3**

```
### Part (c) ###
### YOUR CODE HERE ###
## Hint: Call get_features_from_layer() and use the 'features' attribute of the SaveFeatures class

# Increase the batch size to process data in larger chunks
batch_size = 256  # Increased batch size for faster processing
trainloader = torch.utils.data.DataLoader(training_data, batch_size=batch_size, shuffle=True, num_work

# Function to extract features from a specific layer
def extract_features_subset(model, data_loader, device, target_layer, max_samples=5000):
    # Hook to save the features from the target layer
    features_saver = SaveFeatures(target_layer)

    features = []
    labels = []
    total_samples = 0

    model.eval()  # Set the model to evaluation mode
    with torch.no_grad():  # No gradient needed for inference
        for images, targets in tqdm.tqdm(data_loader, total=len(data_loader), desc="Extracting feature
            images, targets = images.to(device), targets.to(device)

            # Forward pass through the network
            _ = model(images)

            # Collect the extracted features and labels
            features.append(features_saver.features)
            labels.append(targets.cpu().numpy())

            total_samples += images.size(0)
            if total_samples >= max_samples:
                break  # Stop after collecting the desired number of samples

    # Remove the hook after extraction is complete
    features_saver.remove()

    # Concatenate features and labels (truncate if necessary)
    features = np.concatenate(features, axis=0)[:max_samples]
    labels = np.concatenate(labels, axis=0)[:max_samples]

    return features, labels

# Extract features from the third convolutional layer (conv3) for a subset of the data
conv3_layer = net.conv3
features, labels = extract_features_subset(net, trainloader, device, conv3_layer, max_samples=5000)

# Flatten the features from conv3 (if necessary, depends on layer output dimensions)
```

```
features = features.reshape(features.shape[0], -1)

# Normalize the features for better t-SNE performance
features = features.astype('float32') / np.linalg.norm(features, axis=1, keepdims=True)

# Apply PCA to reduce dimensionality before t-SNE for faster computation
pca = PCA(n_components=50)
features_pca = pca.fit_transform(features)

# Apply t-SNE to the PCA-reduced features
tsne = TSNE(n_components=2, random_state=42, verbose=1)
tsne_results = tsne.fit_transform(features_pca)

# Plot the t-SNE results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=labels, cmap='tab10', s=10)
plt.colorbar(scatter, ticks=range(10))
plt.title('t-SNE Visualization of Features from Conv3 Layer (PCA Reduced)')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.show()
```

4. **Q4.4**

```
### Part (d) ###
### YOUR CODE HERE ###

# Extract features from both linear1 and linear2 layers
# Hook class to save the output of the layer


# Function to extract features from a specific layer
def extract_features_from_layers(model, data_loader, device, target_layers, max_samples=5000):
    # Register the hooks on the desired layers
    feature_savers = [SaveFeatures(layer) for layer in target_layers]

    features = []
    labels = []
    total_samples = 0

    model.eval()  # Set the model to evaluation mode
    with torch.no_grad():  # No gradient needed for inference
        for images, targets in tqdm.tqdm(data_loader, total=len(data_loader), desc="Extracting feature
            images, targets = images.to(device), targets.to(device)

            # Forward pass through the network
            _ = model(images)

            # Collect the extracted features and labels
            layer_features = [saver.features for saver in feature_savers]
            features.append(np.concatenate(layer_features, axis=1))  # Concatenate the features from b
            labels.append(targets.cpu().numpy())

            total_samples += images.size(0)
            if total_samples >= max_samples:
                break  # Stop after collecting the desired number of samples

    # Remove the hooks after extraction is complete
    for saver in feature_savers:
        saver.remove()

    # Concatenate features and labels (truncate if necessary)
    features = np.concatenate(features, axis=0)[:max_samples]
    labels = np.concatenate(labels, axis=0)[:max_samples]

    return features, labels

# Extract features from the first and second linear layers (linear1 and linear2)
target_layers = [net.linear1, net.linear2]
features, labels = extract_features_from_layers(net, trainloader, device, target_layers, max_samples=5
# Flatten the features from the layers (if necessary)
features = features.reshape(features.shape[0], -1)

# Normalize the features for better t-SNE performance
features = features.astype('float32') / np.linalg.norm(features, axis=1, keepdims=True)

# Apply PCA to reduce dimensionality before t-SNE for faster computation
pca = PCA(n_components=50)
features_pca = pca.fit_transform(features)

# Apply t-SNE to the PCA-reduced features
tsne = TSNE(n_components=2, random_state=42, verbose=1)
tsne_results = tsne.fit_transform(features_pca)

# Plot the t-SNE results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=labels, cmap='tab10', s=10)
plt.colorbar(scatter, ticks=range(10))
plt.title('t-SNE Visualization of Features from Linear Layers (PCA Reduced)')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.show()
```