

This homework is due at 11 PM on February 20, 2026.

Submission Format: Your homework submission should consist of a single PDF file that contains all of your answers (any handwritten answers should be scanned), as well as a printout of your completed Google Colab notebook(s).

1. Gradients, Jacobians, and Hessians

The *gradient* of a scalar-valued function $g: \mathbb{R}^n \rightarrow \mathbb{R}$ is the column vector of length n , denoted as ∇g , containing the derivatives of components of g with respect to the input variables:

$$(\nabla g(\vec{x}))_i = \frac{\partial g}{\partial x_i}(\vec{x}), \quad i = 1, \dots, n. \quad (1)$$

The *Hessian* of a scalar-valued function $g: \mathbb{R}^n \rightarrow \mathbb{R}$ is the $n \times n$ matrix, denoted as $\nabla^2 g$, containing the second derivatives of components of g with respect to the input variables:

$$(\nabla^2 g(\vec{x}))_{ij} = \frac{\partial^2 g}{\partial x_i \partial x_j}(\vec{x}), \quad i = 1, \dots, n, \quad j = 1, \dots, n. \quad (2)$$

The *Jacobian* of a vector-valued function $\vec{g}: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is the $m \times n$ matrix, denoted as $D\vec{g}$, containing the derivatives of components of \vec{g} with respect to the input variables:

$$(D\vec{g}(\vec{x}))_{ij} = \frac{\partial g_i}{\partial x_j}(\vec{x}), \quad i = 1, \dots, m, \quad j = 1, \dots, n. \quad (3)$$

For the remainder of the class, we will repeatedly have to take gradients, Hessians and Jacobians of functions we are trying to optimize. This exercise serves as a warm up for future problems.

For the first two parts, suppose $A \in \mathbb{R}^{n \times n}$ is a square matrix whose entries are denoted a_{ij} and whose rows are denoted $\vec{a}_1^\top, \dots, \vec{a}_n^\top$, and $\vec{b} \in \mathbb{R}^n$ is a vector whose entries are denoted b_i .

(a) Compute the Jacobians for the following functions.

- i. $\vec{g}(\vec{x}) = A\vec{x}$.
- ii. $\vec{g}(\vec{x}) = f(\vec{x})\vec{x}$ where $f: \mathbb{R}^n \mapsto \mathbb{R}$ is differentiable.
- iii. $\vec{g}(\vec{x}) = f(A\vec{x} + \vec{b})\vec{x}$ where $f: \mathbb{R}^n \mapsto \mathbb{R}$ is differentiable.

(b) Compute the gradients and Hessians for the following functions.

- i. $g_1(\vec{x}) = \vec{x}^\top A\vec{x}$.
- ii. $g_2(\vec{x}) = \|\vec{x}\|_2^2$.
- iii. $g_3(\vec{x}) = g_2(A\vec{x} - \vec{b}) = \|A\vec{x} - \vec{b}\|_2^2$. (Use the chain rule and the Jacobians computed in part 1(a).)
- iv. $g_4(\vec{x}) = \log(\sum_{i=1}^n e^{x_i})$.
- v. $g_5(\vec{x}) = g_4(A\vec{x} - \vec{b}) = \log(\sum_{i=1}^n e^{\vec{a}_i^\top \vec{x} - b_i})$. (Use the chain rule and the Jacobians computed in part 1(a); you can use the gradient ∇g_4 and Hessian $\nabla^2 g_4$ in your answer without having to rewrite it.)
- vi. $g_6(\vec{x}) = e^{\|\vec{x}\|_2^2} = e^{g_2(\vec{x})}$. (Use the chain rule and the Jacobians computed in part 1(a).)
- vii. $g_7(\vec{x}) = e^{\|A\vec{x} - \vec{b}\|_2^2} = g_6(A\vec{x} - \vec{b})$. (Use the chain rule and the Jacobians computed in part 1(a); you can use the gradient ∇g_6 and Hessian $\nabla^2 g_6$ in your answer without having to rewrite it.)

Consider the case now where all vectors and matrices above are scalar; do your answers above make sense?
(No need to answer this in your submission.)

- (c) Plot/hand-draw the level sets of the following functions:

i. $g(x_1, x_2) = \frac{x_1^2}{4} + \frac{x_2^2}{9}$

ii. $g(x_1, x_2) = x_1 x_2$

Also point out the gradient directions in the level-set diagram. Additionally, compute the first and second order Taylor series approximation around the point $(1, 1)$ for each function and comment on how accurately they approximate the true function.

2. Neural Networks and Backpropagation

Neural networks are parametric functions that have been widely used to fit complex patterns in vision and natural languages. Given some training data of the form (\vec{x}_i, y_i) , a neural network \mathcal{N} is trained to minimize a *loss function* on the data. This is often done using *gradient descent*, an optimization method we will cover later in this class. Gradient descent requires us to compute the gradients of the loss function with respect to the parameters of the neural network. In practice, computational frameworks for neural networks compute the gradients automatically and efficiently via *back-propagation*, which uses the chain rule to recursively compute the gradients of the loss function. In this problem, we study a toy neural network trained on a single data point (\vec{x}, y) .

In particular, consider the following simplified three-layer neural network \mathcal{N} , representing a map from \mathbb{R}^d to \mathbb{R} whose parameters are $(\vec{w}_1, w_2, w_3) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$:

$$p_1 = \vec{w}_1^\top \vec{x} \quad (4)$$

$$h_1 = \sigma(p_1) \quad (5)$$

$$p_2 = w_2 h_1 \quad (6)$$

$$h_2 = \sigma(p_2) \quad (7)$$

$$z = w_3 h_2, \quad (8)$$

where $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ is a nonlinear function (also called the “activation function”), whose derivative is denoted by $\sigma': \mathbb{R} \rightarrow \mathbb{R}$.

We want the output of the network $z = \mathcal{N}(\vec{x})$ to match the true label y . A natural choice of the loss function encouraging this behavior is the squared loss:

$$L(y, z) \doteq \frac{1}{2}(y - z)^2. \quad (9)$$

In the parts that follow, we will compute the derivative of L with respect to the parameters \vec{w}_1, w_2, w_3 .

- (a) Compute the following gradients and partial derivatives sequentially from left-to-right:

$$\frac{\partial L}{\partial z}, \quad \frac{\partial L}{\partial w_3}, \quad \frac{\partial L}{\partial h_2}, \quad \frac{\partial L}{\partial p_2}, \quad \frac{\partial L}{\partial w_2}, \quad \frac{\partial L}{\partial h_1}, \quad \frac{\partial L}{\partial p_1}, \quad \nabla_{\vec{w}_1} L, \quad \nabla_{\vec{x}} L. \quad (10)$$

Here $\nabla_{\vec{x}} L$ is the gradient whose entries are the derivatives of L with respect to the entries of \vec{x} , etc. We compute the first 4 derivatives for you.

$$\frac{\partial L}{\partial z} = z - y, \quad \frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z} h_2, \quad \frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial z} w_3, \quad \frac{\partial L}{\partial p_2} = \frac{\partial L}{\partial h_2} \sigma'(p_2) \quad (11)$$

Note how $\frac{\partial L}{\partial w_3}$ can be calculated using $\frac{\partial L}{\partial z}$ and $\frac{\partial L}{\partial p_2}$ can be calculated using $\frac{\partial L}{\partial h_2}$. In numerical computation, the result of $\frac{\partial L}{\partial z}$ and $\frac{\partial L}{\partial h_2}$ can thus be reused. This technique of saving computations for calculating the derivatives of a neural network is called *back-propagation*.

Use the chain rule to calculate the 5 remaining derivatives $\frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial h_1}, \frac{\partial L}{\partial p_1}, \nabla_{\vec{w}_1} L$, and $\nabla_{\vec{x}} L$.

- (b) Now suppose that Equation (8) is written as

$$z' = w_3 h_2 + h_1. \quad (12)$$

That is, we define a new neural network \mathcal{N}' with parameters $(\vec{w}_1, w_2, w_3) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$ as follows.

$$p_1 = \vec{w}_1^\top \vec{x} \quad (13)$$

$$h_1 = \sigma(p_1) \quad (14)$$

$$p_2 = w_2 h_1 \quad (15)$$

$$h_2 = \sigma(p_2) \quad (16)$$

$$z' = w_3 h_2 + h_1. \quad (17)$$

This introduces a change in the network architecture, called the *skip connection*.

Again, compute the following gradients and partial derivatives with respect to the loss function $L(y, z') = \frac{1}{2}(y - z')^2$:

$$\frac{\partial L}{\partial z'}, \quad \frac{\partial L}{\partial w_3}, \quad \frac{\partial L}{\partial h_2}, \quad \frac{\partial L}{\partial p_2}, \quad \frac{\partial L}{\partial w_2}, \quad \frac{\partial L}{\partial h_1}, \quad \frac{\partial L}{\partial p_1}, \quad \nabla_{\vec{w}_1} L, \quad \nabla_{\vec{x}} L. \quad (18)$$

We compute the first 4 derivatives for you.

$$\frac{\partial L}{\partial z'} = z' - y, \quad \frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z'} h_2, \quad \frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial z'} w_3, \quad \frac{\partial L}{\partial p_2} = \frac{\partial L}{\partial h_2} \sigma'(p_2). \quad (19)$$

Use the chain rule to calculate the 5 remaining derivatives $\frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial h_1}, \frac{\partial L}{\partial p_1}, \nabla_{\vec{w}_1} L$, and $\nabla_{\vec{x}} L$.

- (c) In optimizing a neural network using gradient descent, we need the gradient of the loss function with respect to the parameters of the network. Please express $\frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_2}$, and $\nabla_{\vec{w}_1} L$ for \mathcal{N} and \mathcal{N}' respectively with no dependence on partial derivatives of other variables. We compute $\frac{\partial L}{\partial w_3}$ for you, as follows.

- For \mathcal{N} , we have $\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z} h_2 = (z - y)h_2$.
- For \mathcal{N}' , we have $\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z'} h_2 = (z' - y)h_2$.

Express the remaining derivatives $\frac{\partial L}{\partial w_2}$ and $\nabla_{\vec{w}_1} L$ within \mathcal{N} and \mathcal{N}' .

- (d) Many activation functions σ have the property that $\sigma' \leq 1$. For example, the sigmoid function $\sigma(p) = \frac{1}{1+e^{-p}}$ is sometimes used as an activation function. Its derivative, $\sigma'(p) = \frac{e^{-p}}{(1+e^{-p})^2}$ has the range $(0, 1/4]$. That is, $\sigma' < 1$. Consider the case when σ' is much smaller than 1, such that $\sigma'(p)\sigma'(q) \approx 0$ for any p, q , but $\sigma'(p) \not\approx 0$ for any p . Consider the derivatives $\frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_2}$, and $\nabla_{\vec{w}_1} L$ within the neural network \mathcal{N} ; with the above approximations, which of them will approximately be zero? Also answer this question for the neural network \mathcal{N}' .

NOTE: Some of the above gradients will indeed be approximately zero, and this is called the *vanishing gradient* problem in deep learning.

3. PCA and Senate Voting Data

In this problem, we consider a matrix of senate voting data, which we manipulate in Python. The data is contained in a $n \times d$ data matrix X , where each row corresponds to a senator and each column to a bill. Each entry of X is either 1, -1 or 0, depending on whether the senator voted for the bill, against the bill, or abstained, respectively. Please compute your answers using the Google Colab `senator_pca.ipynb`. The sub-parts of this problem can be answered in the notebook itself in the space provided and can be submitted as an attachment to this PDF using the "File > Print > Download as PDF" feature that Google Colab supports. To use Google Colab, you need to make your own private copy of the starter code. Please follow these steps:

Open the Link: Click this [link](#). This will open the Google Colab notebook in "read-only" mode.

Make Your Own Copy: Do not try to type in the notebook yet. Instead, go to the top menu and click File > Save a copy in Drive.

Start Coding: A new tab will open with your personal copy of the notebook and you can safely begin writing your code.

- (a) Suppose we want to assign a *score* to each senator based on their voting pattern, and then observe the empirical variance of these scores. To describe this, let us choose a $\vec{a} \in \mathbb{R}^d$ and a scalar $b \in \mathbb{R}$. We define the score for senator i as:

$$f(\vec{x}_i, \vec{a}, b) = \vec{x}_i^\top \vec{a} + b, \quad i = 1, 2, \dots, n. \quad (20)$$

Note that \vec{x}_i^\top denotes the i^{th} row of X and is a row vector of length d , as in the previous problem.

Let us denote by $\vec{z} = f(X, \vec{a}, b)$ the column vector of length n obtained by stacking the scores for each senator. Then

$$\vec{z} = f(X, \vec{a}, b) = X\vec{a} + b\vec{1} \in \mathbb{R}^n \quad (21)$$

where $\vec{1}$ is a vector with all entries equal to 1. Let us denote the mean value of \vec{z} by $\mu(\vec{z}) = \frac{1}{n}\vec{1}^\top \vec{z}$. Let $\vec{\mu}(X) \in \mathbb{R}^d$ denote the vector containing the mean of each column of X . Then

$$\mu(\vec{z}) = \frac{1}{n} \sum_{i=1}^n z_i \quad (22)$$

$$= \frac{1}{n} \sum_{i=1}^n (\vec{a}^\top \vec{x}_i + b) \quad (23)$$

$$= \vec{a}^\top \left(\frac{1}{n} \sum_{i=1}^n \vec{x}_i \right) + b \quad (24)$$

$$= \vec{a}^\top \vec{\mu}(X) + b \quad (25)$$

The empirical variance of the scores can then be obtained as

$$\sigma^2(\vec{z}) = \frac{1}{n} (\vec{z} - \mu(\vec{z})\vec{1})^\top (\vec{z} - \mu(\vec{z})\vec{1}) \quad (26)$$

$$= \frac{1}{n} ((X\vec{a} + b\vec{1}) - (\vec{a}^\top \vec{\mu}(X) + b)\vec{1})^\top ((X\vec{a} + b\vec{1}) - (\vec{a}^\top \vec{\mu}(X) + b)\vec{1}) \quad (27)$$

$$= \frac{1}{n} (X\vec{a} + b\vec{1} - (\vec{a}^\top \vec{\mu}(X))\vec{1} - b\vec{1})^\top (X\vec{a} + b\vec{1} - (\vec{a}^\top \vec{\mu}(X))\vec{1} - b\vec{1}) \quad (28)$$

$$= \frac{1}{n} (X\vec{a} - (\vec{a}^\top \vec{\mu}(X))\vec{1})^\top (X\vec{a} - (\vec{a}^\top \vec{\mu}(X))\vec{1}) \quad (29)$$

$$= \frac{1}{n} (X\vec{a} - \vec{1}\vec{\mu}(X)^\top \vec{a})^\top (X\vec{a} - \vec{1}\vec{\mu}(X)^\top \vec{a}) \quad (30)$$

$$= \frac{1}{n} ((X - \vec{1}\vec{\mu}(X)^\top) \vec{a})^\top ((X - \vec{1}\vec{\mu}(X)^\top) \vec{a}) \quad (31)$$

$$= \frac{1}{n} \vec{a}^\top (X - \vec{1}\vec{\mu}(X)^\top)^\top (X - \vec{1}\vec{\mu}(X)^\top) \vec{a}. \quad (32)$$

Note that this variance is therefore a function of the “centered” data matrix $X - \vec{1}\vec{\mu}(X)^\top$ in which the mean of each column is zero. It also does not depend on b .

For the remainder of this problem, we assume that the data has been pre-centered (i.e., $\vec{\mu}(X) = \vec{0}$); note that this has been pre-computed for you in the code notebook. Assume also that $b = 0$, so that $\mu(\vec{z}) = 0$.

Defining $f(X, \vec{a}) \doteq f(X, \vec{a}, 0)$ and replacing \vec{z} with $f(X, \vec{a})$, we can then write the simpler empirical variance formula

$$\sigma^2(f(X, \vec{a})) = \frac{1}{n} \vec{a}^\top X^\top X \vec{a}. \quad (33)$$

Suppose we restrict \vec{a} to have unit-norm. In the provided code, find the maximum empirical variance $\sigma^2(f(X, \vec{a}))$ over all unit-norm \vec{a} , and find the \vec{a} that maximizes it.

- (b) We next consider party affiliation as a predictor for how a senator will vote. Follow the instructions in the notebook to compute the mean voting vector for each party and relate it to the direction of maximum variance.
- (c) Recall from problem 1 that given a vector $\vec{z} = X\vec{u}$ (i.e., the vector of scalar projections of each row of X along \vec{u}), we can compute its empirical variance as

$$\sigma^2(\vec{z}) = \vec{u}^\top \Sigma \vec{u}, \quad (34)$$

where $\Sigma(X) = \frac{X^\top X}{n}$ is the empirical covariance matrix of X . We will show in a future homework problem that the variance along each principal component \vec{a}_i is precisely its corresponding eigenvalue of $\Sigma(X)$, i.e., $\lambda_i\{\Sigma(X)\}$. (For now, just note that this fact should make intuitive sense, since PCA is searching for directions of maximum variance of the data, and these occur along the covariance matrix’s eigenvectors.) In the Notebook, compute the sum of the variance along \vec{a}_1 and \vec{a}_2 and plot the data projected on the \vec{a}_1 - \vec{a}_2 plane.

- (d) Suppose we want to find the bills that are most and least contentious — i.e., those that have high variability in senators’ votes, and those for which voting was almost unanimous. Follow the instructions in the notebook to compute a measure of “contentiousness” for each bill, plot the vote counts for exemplar bills, and comment on the voting trends.
- (e) Suppose we want to infer the political affiliations of two senators whose voting records are known to us. Follow the instructions in the notebook to infer the political affiliation of the Green and Grey colored senators using PCA.
- (f) Finally, we can use the defined score $f(X, \vec{a}, b)$, computed along the first principal component \vec{a}_1 to classify the most and least “extreme” senators based on their voting record. Follow the instructions in the Notebook to compute these scores and comment on their relationship to partisan affiliation.

4. SVD Transformation

In this problem we will interpret the linear map defined by matrix $A \in \mathbb{R}^{n \times n}$ by looking at its singular value decomposition, $A = UDV^\top$. Recall that here $U, D, V \in \mathbb{R}^{n \times n}$ and U, V are orthonormal matrices while D is a diagonal matrix. We will first look at how V^\top, D and U each separately transform the unit circle $C = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$ and then look at their effect as a whole. This problem has an associated Google Colab, `svd_transformation.ipynb` that contains parts (b, c, d, e) of the problem. These sub-parts can be answered in the space provided in the notebook itself and can be submitted as an attachment to the solution PDF using the "File > Print > Download as PDF" feature that Google Colab supports. To use Google Colab, you need to make your own private copy of the starter code. Please follow these steps:

Open the Link: Click this [link](#). This will open the Google Colab notebook in "read-only" mode.

Make Your Own Copy: Do not try to type in the notebook yet. Instead, go to the top menu and click File > Save a copy in Drive.

Start Coding: A new tab will open with your personal copy of the notebook and you can safely begin writing your code.

- (a) Let $\vec{x} \in \mathbb{R}^n$ and $\vec{z} = V^\top \vec{x}$. Show that $\vec{x} = \sum_{i=1}^n z_i \vec{v}_i$, where $\vec{z} = [z_1 \ \dots \ z_n]^\top$ and $V = [\vec{v}_1 \ \dots \ \vec{v}_n]$. This shows that \vec{z} is the vector of coordinates that represents \vec{x} in the basis defined by the columns of V .

For the rest of the problem we restrict ourselves to the case where $A \in \mathbb{R}^{2 \times 2}$ and move to the notebook.

5. Homework Process

With whom did you work on this homework? List the names and SIDs of your group members.

NOTE: If you didn't work with anyone, you can put "none" as your answer.