Homework 1 – Deep Learning (CS/DS541, Whitehill, Spring 2023)

1. **Python and Numpy Warm-up Exercises** This part of the homework is intended to help you review your linear algebra and learn (or refresh your understanding of) how to implement linear algebraic and statistical operations in Python using `numpy` (to which we refer in the code below as `np`). For each of the problems below, write a method (e.g., `problem_1a`) that returns the answer for the corresponding problem.

   In all problems, you may assume that the the dimensions of the matrices and/or vectors that are given as input are compatible for the requested mathematical operations.

   Note 1: In mathematical notation we usually start indices with $j = 1$. However, in `numpy` (and many other programming settings), it is more natural to use 0-based array indexing. When answering the questions below, do not worry about "translating" from 1-based to 0-based indexes. For example, if the (i, j)th element of some matrix is requested, you can simply write `A[i,j]`.

   Note 2: To represent and manipulate vectors and matrices, please use `numpy`'s `array` class (*not* the `matrix` class).

   Note 3: While the difference between a row vector and a column vector is important when doing math, `numpy` does not care about this difference *as long as the array is 1-D*. This means, for example, that if you want to compute the inner product between two vectors `x` and `y`, you can just write `x.dot(y)` without needing to transpose the `x`. If `x` and `y` are 2-D arrays, however, then it *does* matter whether they are row-vectors or column-vectors, and hence you might need to transpose accordingly.

   (a) Given two matrices $\mathbf{A}$ and $\mathbf{B}$, compute and return an expression for $\mathbf{A} + \mathbf{B}$. [**0 pts**]
   Answer: While it is completely valid to use `np.add(A, B)`, this is unnecessarily verbose; you really should make use of the "syntactic sugar" provided by Python's/`numpy`'s operator overloading and just write: `A + B`. Similarly, you should use the more compact (and arguably more elegant) notation for the rest of the questions as well.

   (b) Given matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, compute and return $\mathbf{AB} - \mathbf{C}$ (i.e., right-multiply matrix $\mathbf{A}$ by matrix $\mathbf{B}$, and then subtract $\mathbf{C}$). Use `dot` or `np.dot`. [**1 pts**]

   (c) Given matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, return $\mathbf{A} \odot \mathbf{B} + \mathbf{C}^\top$, where $\odot$ represents the element-wise (Hadamard) product and $\top$ represents matrix transpose. In `numpy`, the element-wise product is obtained simply with `*`. [**1 pts**]

   (d) Given column vectors $\mathbf{x}$ and $\mathbf{y}$, compute the inner product of $\mathbf{x}$ and $\mathbf{y}$ (i.e., $\mathbf{x}^\top \mathbf{y}$). [**1 pts**]

   (e) Given square matrix $\mathbf{A}$ and column vector $\mathbf{x}$, use `np.linalg.solve` to compute $\mathbf{A}^{-1}\mathbf{x}$. Do **not** explicitly calculate the matrix inverse itself (e.g., `np.linalg.inv`, `A ** -1`) because this is numerically unstable (and yes, it can sometimes make a big difference!). [**2 pts**]

   (f) Given matrix $\mathbf{A}$ and integer i, return the sum of all the entries in the ith row *whose column index is even*, i.e., $\sum_{j:j \text{ is even}} \mathbf{A}_{ij}$. Do **not** use a loop, which in Python can be very slow. Instead use the `np.sum` function. [**2 pts**]

   (g) Given matrix $\mathbf{A}$ and scalars c, d, compute the arithmetic mean over all entries of A that are between c and d (inclusive). In other words, if $\mathcal{S} = \{(i, j) : c \leq \mathbf{A}_{ij} \leq d\}$, then compute $\frac{1}{|\mathcal{S}|} \sum_{(i,j) \in \mathcal{S}} \mathbf{A}_{ij}$. Use `np.nonzero` along with `np.mean`. [**2 pts**]

   (h) Given an $(n \times n)$ matrix $\mathbf{A}$ and integer k, return an $(n \times k)$ matrix containing the right-eigenvectors of $\mathbf{A}$ corresponding to the k eigenvalues of $\mathbf{A}$ with the largest absolute value. Use `np.linalg.eig`. [**3 pts**]

   (i) Given a column vector (with n components) $\mathbf{x}$, an integer k, and positive scalars m, s, return an $(n \times k)$ matrix, each of whose columns is a sample from multidimensional Gaussian distribution $\mathcal{N}(\mathbf{x} + m\mathbf{z}, s\mathbf{I})$, where $\mathbf{z}$ is column vector (with n components) containing all ones and $\mathbf{I}$ is the identity matrix. Use either `np.random.multivariate_normal` or `np.random.randn`. [**3 pts**]

(j) Given a matrix $\mathbf{A}$ with n rows, return a matrix that results from **randomly permuting** the columns (but not the rows) in $\mathbf{A}$. [**2 pts**]

(k) Z-scoring: Given a vector $\mathbf{x}$, return a vector $\mathbf{y}$ such that each $y_i = (x_i - \overline{\mathbf{x}})/\sigma$, where $\overline{\mathbf{x}}$ is the mean (use `np.mean`) of the elements of $\mathbf{x}$ and σ is the standard deviation (use `np.std`). [**2 pts**]

(l) Given an n-vector $\mathbf{x}$ and a non-negative integer k, return a $n \times k$ matrix consisting of k copies of $\mathbf{x}$. You can use numpy methods such as `np.newaxis`, `np.atleast_2d`, and/or `np.repeat`. [**2 pts**]

(m) Given a $k \times n$ matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \end{bmatrix}$ and a $k \times m$ matrix $\mathbf{Y} = \begin{bmatrix} \mathbf{y}^{(1)} & \dots & \mathbf{y}^{(m)} \end{bmatrix}$, compute an $n \times m$ matrix $\mathbf{D} = \begin{bmatrix} d_{11} & \dots & d_{1m} \\ & \ddots & \\ d_{n1} & \dots & d_{nm} \end{bmatrix}$ consisting of all pairwise L_2 distances $d_{ij} = \|\mathbf{x}^{(i)} - \mathbf{y}^{(j)}\|_2$. In this problem you may **not** use loops. Instead, you can avail yourself of numpy objects & methods such as `np.newaxis`, `np.atleast_3d`, `np.repeat`, `np.swapaxes`, etc. (There are various ways of solving it.) **Hint**: from $\mathbf{X}$ (resp. $\mathbf{Y}$), construct a 3-d matrix that contains multiple copies of each of the vectors in $\mathbf{X}$ (resp. $\mathbf{Y}$); then subtract these 3-d matrices. [**4 pts**]

(n) Given a list of matrices ($\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, $\mathbf{A}^{(3)}$, ...), compute the number of multiplication operations that are necessary in order to multiply these matrices **from left to right** (for the previous example above: $((\mathbf{A}^{(1)}\mathbf{A}^{(2)})\mathbf{A}^{(3)})\dots$) using the standard matrix multiplication approach (**not** anything fancier like Strassen's algorithm). Hint: To access the number of rows or columns of the ith matrix in `matrices`, you can use `matrices[i].shape[0]` or `matrices[i].shape[1]`, respectively. [**4 pts**]

2. **Linear Regression via Analytical Solution**

(a) Train an age regressor that analyzes a ($48 \times 48 = 2304$)-pixel grayscale face image and outputs a real number $\hat{y}$ that estimates how old the person is (in years). Your regressor should be implemented using linear regression. The training and testing data are available here:

- `https://s3.amazonaws.com/jrwprojects/age_regression_Xtr.npy`
- `https://s3.amazonaws.com/jrwprojects/age_regression_ytr.npy`
- `https://s3.amazonaws.com/jrwprojects/age_regression_Xte.npy`
- `https://s3.amazonaws.com/jrwprojects/age_regression_yte.npy`

To get started, see the `train_age_regressor` function in `homework1_template.py`.

Note: you must complete this problem using only linear algebraic operations in `numpy` – you may **not** use any off-the-shelf linear regression software, as that would defeat the purpose.

Compute the optimal weights $\mathbf{w} = (w_1, \dots, w_{2304})$ and optimal bias term b for a linear regression model by deriving the expression for the gradient of the cost function w.r.t. $\mathbf{w}$, setting it to 0, and then solving. Do **not** solve using gradient descent. The cost function is

$$f_{\text{MSE}}(\mathbf{w}, b) = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2$$

where $\hat{y} = g(\mathbf{x}; \mathbf{w}, b) = \mathbf{x}^\top \mathbf{w} + b$ and n is the number of examples in the training set $\mathcal{D}_{\text{tr}} = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}$, each $\mathbf{x}^{(i)} \in \mathbb{R}^{2304}$ and each $y^{(i)} \in \mathbb{R}$. After optimizing $\mathbf{w}$ and b only on the **training set**, compute and report the cost f_{MSE} on the training set $\mathcal{D}_{\text{tr}}$ and (separately) on the testing set $\mathcal{D}_{\text{te}}$. Please report these numbers in the PDF file. [**15 pts**]

3. **Proofs/Derivations** For the proofs, please create a PDF (which you can generate using LaTeX, or, if you prefer, a scanned copy of your **legible** handwriting).

(a) Let $\nabla_{\mathbf{x}} f(\mathbf{x})$ represent the column vector containing all the partial derivatives of f w.r.t. $\mathbf{x}$, i.e.,

$$\nabla_{\mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

For any two column vectors $\mathbf{x}, \mathbf{a} \in \mathbb{R}^n$, prove that

$$\nabla_{\mathbf{x}} \left(\mathbf{x}^\top \mathbf{a}\right) = \nabla_{\mathbf{x}} \left(\mathbf{a}^\top \mathbf{x}\right) = \mathbf{a}$$

Required approach: differentiate w.r.t. each element of $\mathbf{x}$ (i.e., $x_1, \ldots, x_n$), and then gather all the n partial derivatives into a column vector. **[4 pts]**

(b) Prove that

$$\nabla_{\mathbf{x}} \left(\mathbf{x}^\top \mathbf{A} \mathbf{x}\right) = (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$$

for any column vector $\mathbf{x} \in \mathbb{R}^n$ and any $n \times n$ matrix $\mathbf{A}$. Required approach: differentiate w.r.t. each element of $\mathbf{x}$ (i.e., $x_1, \ldots, x_n$), and then gather all the n partial derivatives into a column vector. **[6 pts]**

(c) Based on the theorem above, prove that

$$\nabla_{\mathbf{x}} \left(\mathbf{x}^\top \mathbf{A} \mathbf{x}\right) = 2\mathbf{A}\mathbf{x}$$

for any column vector $\mathbf{x} \in \mathbb{R}^n$ and any symmetric $n \times n$ matrix $\mathbf{A}$. **[2 pts]**

(d) Based on the theorems above, prove that

$$\nabla_{\mathbf{x}} \left[(\mathbf{A}\mathbf{x} + \mathbf{b})^\top (\mathbf{A}\mathbf{x} + \mathbf{b})\right] = 2\mathbf{A}^\top (\mathbf{A}\mathbf{x} + \mathbf{b})$$

for any column vector $\mathbf{x} \in \mathbb{R}^n$, any symmetric $n \times n$ matrix $\mathbf{A}$, and any constant column vector $\mathbf{b} \in \mathbb{R}^n$. **[4 pts]**

Submission: Create a Zip file containing both your Python and PDF files, and then submit on Canvas. If you are working as part of a group, then only **one** member of your group should submit (but make sure you have already signed up in a pre-allocated team for Homework 1 on Canvas).
Teamwork: You may complete this homework assignment either individually or in a team (up to 2 people unless you have special permission from the instructor).