# Homework 1 – Deep Learning (CS/DS541, Murai, Fall 2022)

(Adapted from Prof. Whitehill)

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 are compatible for the requested mathematical operations. **Note**: Throughout the assignment, please use `np.array`, not `np.matrix`.

   (a) Given 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 the array multiplication operator `@`. [ **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 square matrix $\mathbf{A}$ and row vector $\mathbf{x}$, use `np.linalg.solve` to compute $\mathbf{x}\mathbf{A}^{-1}$. Hint: $\mathbf{AB} = (\mathbf{B}^\top \mathbf{A}^\top)^\top$. [ **3 pts** ]

   (g) Given matrix $\mathbf{A}$ and integer $i$, return the sum of all the entries in the $i$th 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** ]

   (h) 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** ]

   (i) 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 $k$ largest eigenvalues of $\mathbf{A}$. Use `np.linalg.eig`. [ **3 pts** ]

   (j) 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** ]

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

   (l) 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 $\sigma$ is the standard deviation (use `np.std`). [ **2 pts** ]

   (m) 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** ]

(n) Given an $m \times n$ matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \end{bmatrix}$, compute an $n \times n$ matrix $\mathbf{D} = \begin{bmatrix} d_{11} & \dots & d_{1n} \\ & \ddots & \\ d_{n1} & \dots & d_{nn} \end{bmatrix}$ consisting of all pairwise $L_2$ distances $d_{ij} = \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|_2$. 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**: construct two different 3-d arrays that each contain $n$ copies of each of the $n$ vectors in $\mathbf{X}$; then subtract them. **[ 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 $\mathbf{w} = (w_1, \dots, w_{2304})$ 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

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

where $\hat{y} = g(\mathbf{x}; \mathbf{w}) = \mathbf{x}^\top \mathbf{w}$ 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 $y^{(i)} \in \mathbb{R}$. Note that this simple regression model does not include a bias term (which we will add later); correspondingly, please do **not** include one in your own implementation. After optimizing $\mathbf{w}$ only on the **training set**, compute and report the cost $f_{\text{MSE}}$ 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. **[ 10 pts ]**

3. **Probability Distributions**

(a) **Estimating the Parameters of a Probability Distribution**: Plot the empirical probability distribution of the data in the included `PoissonX.npy` (you can use `matplotlib.pyplot.hist` with `density=True`). Then, using `scipy.stats.poisson`, plot the probability distributions of a Poisson random variable with (alternative) rate parameters of 2.5, 3.1, 3.7, and 4.3. (Make sure to include these plots in your homework submission.) Based on visual inspection of the idealized distributions with the empirical distribution, report which of the possible parameter values is most consistent with the data. (Later on, we will formalize the parameter estimation process with Maximum Likelihood Estimation.) **[ 4 pts ]**

(b) **Conditional Probability Distributions to Represent the Uncertainty of Functions**: Consider the conditional probability distribution

$$P(y \mid x) = \mathcal{N}\left(\mu = x^2, \sigma^2 = \left(2 - \frac{1}{1 + e^{-x^2}}\right)^2\right)$$

(where $\mathcal{N}$ is the Normal distribution with a specified mean and variance) representing the uncertainty of the $y$-value associated with any given value $x$.

i. For which values of $x$ – those with small magnitude, or those with large magnitude – does the corresponding value of $y$ tend to be larger? [ **1 pt** ]

ii. For which values of $x$ – those with small magnitude, or those with large magnitude – does the *uncertainty* in the corresponding value of $y$ tend to be larger? [ **1 pt** ]

4. **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

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

   Hint: differentiate w.r.t. each element of $\mathbf{x}$, and then gather the 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 $\mathbf{A}$. [ **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.
**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).