Homework 1 – Deep Learning (CS/DS541, Murai, Fall 2023)

(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 (freebie!): 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**]
   Answer: `return A @ B - C`

   (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**]
   Answer: `return A * B + C.T`

   (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**]
   Answer: `return x.dot(y)`

   (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**]
   Answer: `return np.linalg.solve(A, x)`

   (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**]
   Answer: `return np.linalg.solve(A.T, x).T`

   (g) Given matrix $\mathbf{A}$ and integer i, return the sum of all the entries in the jth column *whose row index is even*, i.e., $\sum_{i:i \text{ is even}} \mathbf{A}_{ij}$. Do **not** use a loop, which in Python can be very slow. Instead use the `np.sum` function. [**2 pts**]
   Answer: `return np.sum(A[0::2,j])`

   (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 $\mathcal{S}=\{(i,j) : c \le \mathbf{A}_{ij} \le d\}$, then compute $\frac{1}{|\mathcal{S}|}\sum_{(i,j)\in\mathcal{S}}\mathbf{A}_{ij}$. Use `np.nonzero` along with `np.mean`. [**2 pts**]
   Answer: `return np.mean(A[np.nonzero((A >= c) * (A <= d))])`

   (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**]
   Answer:
   ```
   l, v = np.linalg.eig(A)
   return v[:, np.argsort(np.abs(l))[-k:]]
   ```

   (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**]
Answer: `return np.random.multivariate_normal(x + m, s * np.eye(x.shape[0]), k).T`

(k) Given a matrix **A** with n rows, return a matrix that results from **randomly permuting** the rows (but not the columns) in **A**. [**2 pts**]
Answer: `return A[np.random.permutation(A.shape[0]), :]`

(l) Z-scoring: Given a vector **x**, return a vector **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 **x** and σ is the standard deviation (use `np.std`). [**2 pts**]
Answer: `return (x - np.mean(x)) / np.std(x)`

(m) Given an n-vector **x** and a non-negative integer k, return a $n \times k$ matrix consisting of k copies of **x**. Use `np.atleast_2d` on **x** and then use `np.repeat`. [**2 pts**]
Answer: `return np.repeat(np.atleast_2d(x), k, axis=1)`

2. **Debugging numpy errors.** For this question ONLY, you can use ChatGPT to debug the code. If you do so, you need to: (i) disclose it in your answer, (ii) indicate if you agree with ChatGPT's initial answer and, if not, (iii) you can describe how you got it to give you the correct answer or you can figure it out by yourself.

(a) Why the code below isn't subtracting each row's minimum element from the respective row? [**2 pts**]

```
X = np.arange(9).reshape(3,3)
row_min = X.min(axis=1)
print(X-row_min)
```

Answer: ChatGPT doesn't answer well questions related to broadcasting. The problem here is that after taking the minimum, the result stored in `row_min` is a 1D array of shape $(3,)$. Since the number of dimensions does not match with X, broadcasting adds one dimension to the left of `row_min`, making it $(1, 3)$, i.e., a row matrix. Since the first dimension doesn't match, the row is replicated 3 times before being subtracted from X. One simple fix is to set `keepdim=True` in the `X.min` function.

(b) Change the code above so as to compute an array **Y** of shape $(3, 3, 3)$ such that:

$$\mathbf{Y}_{i,j,k} = \mathbf{X}_{j,k} - \texttt{row\_min}_i.$$

You **must not** use loops. Explain the steps performed by numpy's broadcasting. [**2 pts**]
Answer: Append the following to the code in item (a):

```
Y=X-row_min[:,np.newaxis,np.newaxis]
```

3. **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://canvas.wpi.edu/files/5766182/download?download_frd=1`
- `https://canvas.wpi.edu/files/5766184/download?download_frd=1`
- `https://canvas.wpi.edu/files/5766181/download?download_frd=1`
- `https://canvas.wpi.edu/files/5766183/download?download_frd=1`

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 $\mathbf{w} = (w_1, \ldots, w_{2304})$ 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

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

where $\hat{y} = g(\mathbf{x}; \mathbf{w}) = \mathbf{x}^\top \mathbf{w}$ and n is the number of examples in the training set $\mathcal{D}_{\mathrm{tr}} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$, 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_{MSE} on the training set $\mathcal{D}_{\mathrm{tr}}$ and (separately) on the testing set $\mathcal{D}_{\mathrm{te}}$. Please report these numbers in the PDF file. [**10 pts**]

Answer: Assuming $\mathbf{X}$ is $k \times n$ and y is of length n, where n is number of training examples and k is number of features (i.e., pixels), then compute: `w = np.linalg.solve(X.dot(X.T), X.T.dot(y))` Then, to obtain the (half) MSE, compute: `loss = 1./(2*X.shape[1]) * np.sum((X.T.dot(w) - y) ** 2)`

4. **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**]
   Answer: $\lambda = 3.7$; this can be verified visually by comparing the histogram of the data in `PoissonX.npy` and the probability density of $\mathrm{Pois}(\lambda = 3.7)$ versus the other values of λ.

   (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**] *Answer*: x values with larger magnitude.

   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**]
   Answer: x values with smaller magnitude.

   iii. (Using the appropriate scipy function) For $x = 1$, what is the probability that a r.v. Y sampled from that distribution is positive? [**1 pt**] *Answer*: 1-0.215 = 0.785

5. **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**]
Answer: For each $k \in \{1, \ldots, n\}$,

$$\frac{\partial}{\partial x_k} \left(\mathbf{x}^\top \mathbf{a}\right) = \frac{\partial}{\partial x_k} \left(\mathbf{a}^\top \mathbf{x}\right) = \frac{\partial}{\partial x_k} \left(\sum_{i=1}^{n} a_i x_i\right) = a_k$$

Therefore,

$$\nabla_{\mathbf{x}}(\mathbf{x}^\top \mathbf{a}) = \nabla_{\mathbf{x}}(\mathbf{a}^\top \mathbf{x}) = \begin{bmatrix} \frac{\partial(\sum_{i=1}^n a_i x_i)}{\partial x_1} \\ \vdots \\ \frac{\partial(\sum_{i=1}^n a_i x_i)}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} = \mathbf{a}$$

(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**]
Answer:

$$\mathbf{x}^\top \mathbf{A} \mathbf{x} = \begin{bmatrix} \sum_i x_i A_{i1} & \ldots & \sum_i x_i A_{in} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \sum_j \sum_i x_i A_{ij} x_j = \sum_i \sum_j x_i A_{ij} x_j$$

In other words, $\mathbf{x}^\top \mathbf{A} \mathbf{x}$ is the sum over a 2-d "grid" of terms. For any $k \in \{1, \ldots, n\}$, $2n - 1$ of the terms in the grid involve x_k: $2(n-1)$ are linear in x_k, and 1 term is quadratic in x_k. We handle these separately when taking the derivative:

$$\frac{\partial(\mathbf{x}^\top \mathbf{A} \mathbf{x})}{\partial x_k} = \frac{\partial}{\partial x_k} \left(\sum_{j \neq k} x_k x_j A_{kj} + \sum_{i \neq k} x_i x_k A_{ik} + x_k x_k A_{kk} \right) \quad (1)$$

$$= \sum_{j \neq k} x_j A_{kj} + \sum_{i \neq k} x_k A_{ik} + 2 x_k A_{kk} \quad (2)$$

$$= \sum_{j \neq k} x_j A_{kj} + \sum_{i \neq k} x_k A_{ik} + x_k A_{kk} + x_k A_{kk} \quad (3)$$

Now we can complete each summation by including the term $x_k A_{kk}$:

$$\frac{\partial(\mathbf{x}^\top \mathbf{A} \mathbf{x})}{\partial x_k} = \sum_j x_j A_{kj} + \sum_i x_i A_{ik} \quad (4)$$

$$= (\mathbf{A}_{k\cdot})\,\mathbf{x} + \left((\mathbf{A}^\top)_{k\cdot}\right)\mathbf{x} \quad (5)$$

where $\mathbf{A}_{k\cdot}$ is the kth row of matrix $\mathbf{A}$. Finally, we obtain

$$\nabla_{\mathbf{x}}(\mathbf{x}^\top \mathbf{A}\mathbf{x}) = \begin{bmatrix} \frac{\partial(\mathbf{x}^\top \mathbf{A}\mathbf{x}))}{\partial x_1} \\ \vdots \\ \frac{\partial(\mathbf{x}^\top \mathbf{A}\mathbf{x}))}{\partial x_n} \end{bmatrix} = \begin{bmatrix} (\mathbf{A}_{1\cdot})\,\mathbf{x} + \left((\mathbf{A}^\top)_{1\cdot}\right)\mathbf{x} \\ \vdots \\ (\mathbf{A}_{n\cdot})\,\mathbf{x} + \left((\mathbf{A}^\top)_{n\cdot}\right)\mathbf{x} \end{bmatrix} = (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$$

(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**]
Answer: Since $\mathbf{A}$ is symmetric, then $\mathbf{A} = \mathbf{A}^\top$, and thus $(\mathbf{A} + \mathbf{A}^\top)\mathbf{x} = 2\mathbf{A}\mathbf{x}$.

(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**]
Answer:

$$\begin{aligned} (\mathbf{A}\mathbf{x} + \mathbf{b})^\top(\mathbf{A}\mathbf{x} + \mathbf{b}) &= \mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x} + \mathbf{x}^\top\mathbf{A}^\top\mathbf{b} + \mathbf{b}^\top\mathbf{A}\mathbf{x} + \mathbf{b}^\top\mathbf{b} && (6) \\ &= \mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x} + \mathbf{x}^\top\mathbf{A}^\top\mathbf{b} + (\mathbf{A}^\top\mathbf{b})\mathbf{x} + \mathbf{b}^\top\mathbf{b} && (7) \end{aligned}$$

Applying the gradient rules derived above (and noting that $\mathbf{A}^\top\mathbf{A}$ is itself a symmetric matrix), we obtain

$$\begin{aligned} \nabla_{\mathbf{x}}\left((\mathbf{A}\mathbf{x} + \mathbf{b})^\top(\mathbf{A}\mathbf{x} + \mathbf{b})\right) &= 2\mathbf{A}^\top\mathbf{A}\mathbf{x} + \mathbf{A}^\top\mathbf{b} + \mathbf{A}^\top\mathbf{b} && (8) \\ &= 2\mathbf{A}^\top\mathbf{A}\mathbf{x} + 2\mathbf{A}^\top\mathbf{b} && (9) \\ &= 2\mathbf{A}^\top(\mathbf{A}\mathbf{x} + \mathbf{b}) && (10) \end{aligned}$$