

Homework 1 – Deep Learning (CS/DS541, Whitehill, Spring 2020)

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 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., `problem1`) that returns the answer for the corresponding problem. Put all your methods in one file called `homework1_WPIUSERNAME.py` (e.g., `homework1_jrwhitehill.py`). See the starter file `homework1_template.py`.

In all problems, you may assume that 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 $\mathbf{A} + \mathbf{B}$. [1 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 `dot` or `np.dot`. [2 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 `*`. [2 pts]
- (d) Given column vectors \mathbf{x} and \mathbf{y} , compute the inner product of \mathbf{x} and \mathbf{y} (i.e., $\mathbf{x}^\top \mathbf{y}$). [2 pts]
- (e) Given matrix \mathbf{A} , return a matrix with the same dimensions as \mathbf{A} but that contains all zeros. Use `np.zeros`. [2 pts]
- (f) 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]
- (g) 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]
- (h) Given square matrix \mathbf{A} and (scalar) α , compute $\mathbf{A} + \alpha \mathbf{I}$, where \mathbf{I} is the identity matrix with the same dimensions as \mathbf{A} . Use `np.eye`. [2 pts]
- (i) Given matrix \mathbf{A} and integers i, j , return the j th column of the i th row of \mathbf{A} , i.e., \mathbf{A}_{ij} . [1 pts]
- (j) 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]
- (k) Given matrix \mathbf{A} and scalars c, d , compute the arithmetic mean over all entries of \mathbf{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`. [3 pts]
- (l) 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]
- (m) Given a n -dimensional column vector \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 an n -dimensional column vector containing all ones and \mathbf{I} is the identity matrix. Use either `np.random.multivariate_normal` or `np.random.randn`. [3 pts]
- (n) 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]

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} and b , 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_{MSE} on the training set \mathcal{D}_{tr} and (separately) on the testing set \mathcal{D}_{te} . [12 pts]

3. **Proofs** 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) Prove that

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

for any two n -dimensional column vectors \mathbf{x}, \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

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

for any n -dimensional column vector \mathbf{x} and any $n \times n$ matrix \mathbf{A} . [8 pts]

- (c) Based on the theorem above, prove that

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

for any n -dimensional column vector \mathbf{x} and any symmetric $n \times n$ matrix \mathbf{A} . [2 pts]

- (d) Based on the theorems above, prove that

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

for any n -dimensional column vector \mathbf{x} , any symmetric $n \times n$ matrix \mathbf{A} , and any constant n -dimensional column vector \mathbf{b} . [4 pts]

Submission: Create a Zip file containing both your Python and PDF files, and then submit on Canvas.

Teamwork: You may complete this homework assignment either individually or in teams up to 2 people.