Homework 1 – Deep Neural Networks (CS525 191N, Whitehill, Spring 2017)

This homework is intended to help you learn (or refresh your understanding of) how to implement linear algebraic operations in Python using `numpy`.

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 the dimensions of the matrices and/or vectors are compatible for the requested mathematical operations.

1. Given matrices $\mathbf{A}$ and $\mathbf{B}$, compute and return an expression for $\mathbf{A} + \mathbf{B}$. [**2 pts**]

2. 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 `numpy.dot`. [**2 pts**]

3. 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**]

4. 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**]

5. Given matrix $\mathbf{A}$, return a matrix with the same dimensions as $\mathbf{A}$ but that contains all zeros. Use `numpy.zeros`. [**2 pts**]

6. Given matrix $\mathbf{A}$, return a vector with the same number of rows as $\mathbf{A}$ but that contains all ones. Use `numpy.ones`. [**2 pts**]

7. Given (invertible) matrix $\mathbf{A}$, compute $\mathbf{A}^{-1}$. [**2 pts**]

   (Now that you know how to compute the inverse of a matrix, you should **almost never need to do so ever again**. The reasons are that (1) in most numerical algorithms, you are not interested in the matrix inverse itself but rather in the inverse **multiplied** by something else (e.g., a vector); and (2) computing the matrix inverse explicitly is numerically unstable.)

8. Given square matrix $\mathbf{A}$ and column vector $\mathbf{x}$, use `numpy.linalg.solve` to compute $\mathbf{A}^{-1}\mathbf{x}$. [**2 pts**]

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

10. 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 `numpy.eye`. [**2 pts**]

11. Given matrix $\mathbf{A}$ and integers i, j, return the jth column of the ith row of $\mathbf{A}$, i.e., $\mathbf{A}_{ij}$. [**2 pts**]

12. Given matrix $\mathbf{A}$ and integer i, return the sum of all the entries in the ith row, i.e., $\sum_j \mathbf{A}_{ij}$. Do **not** use a loop, which in Python is very slow. Instead use the `numpy.sum` function. [**4 pts**]

13. 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 `numpy.nonzero` along with `numpy.mean`. [**4 pts**]

14. 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 `numpy.linalg.eig` to compute eigenvectors. [**4 pts**]

15. 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 `numpy.random.multivariate_normal` or `numpy.random.randn`. [**5 pts**]