Assignment 1: NumPy, Computational Linear Algebra, and Word2Vec

LING 380/780, Fall 2021
Neural Network Models of Linguistic Structure

Assigned: Friday, September 17
Due: Monday, September 27, 11:59 PM
Total Points: 100

This assignment will introduce you to the manipulation of vectors and matrices with Python and give you the opportunity to apply these skills to Word2Vec embeddings by implementing a function that computes analogies. For the assignment, please download all the files in the `homework/homework 1` folder in the Files section of Canvas. You will complete Python programming exercises by filling out function definitions in the `numpy_exercise.py` and `word2vec_exercise.py` files, and you will put written answers in a Markdown[1] file called `assignment1.md`.

1 Linear Algebra and NumPy Exercises

In the first part of the assignment, you will practice using NumPy, a Python library that efficiently implements linear algebra operations.

1.1 Installation

To obtain the software necessary for this assignment, please download and install the Anaconda distribution of Python, available at https://www.anaconda.com/. If you do not have Anaconda installed already, you should choose the latest version of the Individual Edition, which comes with Python 3.8. Anaconda comes with a number of Python packages that are essential to working with neural networks, as well as the `conda` package manager.

If you do not have enough disk space for Anaconda, you can choose to install Miniconda instead, available at https://docs.conda.io/en/latest/miniconda.html. Miniconda is

[1] https://www.markdownguide.org/basic-syntax/

a version of Anaconda that only includes the `conda` package manager. If you choose to use Miniconda or an existing Python installation,[2] please ensure that you have the following installed.

- Python 3.5 or a later version
- The `pip` package manager or `conda` package manager. If you don't have either package manager, please install `pip` by following the instructions at https://pip.pypa.io/en/stable/installation/.
- The NumPy package. If you don't have NumPy installed, please follow the instructions at https://numpy.org/install/.

1.2 Basic Operations (3 Points Each)

In the following exercises, you will read snippets of code and describe what they do in plain English. You may choose to run the code snippets in the Python console, a Python script, or a Jupyter Notebook.[3] You may also consult the NumPy documentation at https://numpy.org/doc/stable/. Each code snippet assumes that all previous code snippets have already been run. Therefore, you must run the code snippets in the same order as they appear in the instructions.

To begin, please import the NumPy package, as follows.

```
import numpy as np
```

By convention, we typically refer to the NumPy package using the alias `np`.

Problem 1. The heart of NumPy is the *array*, a data type that represents vectors and matrices. Please create some arrays using the following code.

```
a = np.array(1)
b = np.array([1, 2, 3])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]],
              [[7, 8, 9], [10, 11, 12]]])
```

What mathematical objects are represented by `a`, `b`, `c`, and `d`?

Problem 2. An important property of an array is its *shape*. Please run the following code in order to inspect the shape of our arrays.

[2] This is not recommended unless you have an existing Python or Anaconda setup that you are already accustomed to using. If you are new to Python, you should install Anaconda, even if your computer (Mac OS, Linux, and other Unix-based operating systems) has a pre-installed version of Python.

[3] https://jupyter.org/

```
print(a.shape)
print(b.shape)
print(c.shape)
print(d.shape)
```

What is the `.shape` of an array?

Problem 3. What do the following lines of code do? In your answer, refer to the lines by their line number (line 1, line 2, etc.).

```
c[0]
c[:, 0]
c[-1]
c[1:]
c[:, np.newaxis]
```

Problem 4. Please run the following lines of code.

```
print(c + c)
print(c - c)
print(c * c)
print(c / c)
```

What do `+`, `-`, `*`, and `/` do?

Problem 5. Please run the following code.

```
print(5 + c)
print(a + c)
print(b + c)
```

What happens when you add a scalar or a vector to a matrix?

Problem 6. Please run the following code.

```
print(c.T)
print(c @ c.T)
```

What does `.T` do? What does `@` do?

Problem 7. Please run the following code.

```
# Create a random array of shape (2, 3, 4)
e = np.random.randint(-10, high=10, size=(2, 3, 4))
print(d @ e)
print((d @ e).shape)
```

What does `@` do when its arguments are 3-dimensional or higher-dimensional arrays?

Problem 8. Please run the following code.

```
print(c.sum())
print(c.sum(axis=0))
print(c.sum(axis=1))
print(c.sum(axis=1, keepdims=True))
```

What does `.sum()` do? What do the `axis` and `keepdims` keyword parameters do?

1.3 Complex Operations (6 Points Each)

Next, you will implement some array operations that are frequently used in machine learning. In each of the problems below, **you must implement the operation described using only one line of code**. Your one line of code cannot exceed 76 characters in width, excluding indentations. For each problem, you must fill out one of the functions in the `numpy_exercise.py` file by replacing `pass` with your own code. Please read the docstring[4] of each function for guidance on what each function should do. You should consult the NumPy documentation and look for operations that will help you implement the functions.

Problem 9. Please implement the `sigmoid` function, which applies sigmoid to each item in an array.

Problem 10. Please implement the `zero_center` function, which takes a matrix and subtracts the mean value from each row.

Problem 11. Please implement the `even_rows` function, which will pick out the rows of an array with even index.

Problem 12. Please implement the `mask` function, which will "mask out" a value from an array, replacing it with another value.

Problem 13. Please implement the `accuracy` function. This function takes a matrix of logit scores (i.e., confidence scores) produced by a multi-class classifier along with a vector of labels, and computes the proportion of examples that have been classified correctly.

1.4 Analysis of Matrix Multiplication (9 Points)

Python is a high-level programming language that is easy to learn and easy to read. The beauty and simplicity of Python comes at a cost, however: Python operations are notoriously slow compared to lower-level languages such as C. Fortunately, NumPy array

[4] https://www.python.org/dev/peps/pep-0257/

operations implement many of the most common yet expensive linear algebra computations using optimized, pre-compiled C code. This means that the more you use NumPy operations, the more efficiently your Python code will run.

In this exercise, you will compare the performance of NumPy's matrix multiplication operator against a pure Python implementation of matrix multiplication.

Problem 14. (6 Points) Please implement the `matmul_pure_python` function, which applies matrix multiplication to its two arguments. Assume that matrices are represented as lists of lists of numbers: each list of numbers is a row, and the full matrix is a list of rows. You may assume that your inputs are always valid matrices.

Problem 15. (3 Points) Please run the function `measure_time` after you have completed Problem 14. This function will generate two random matrices, multiply them using your pure Python implementation of matrix multiplication and using NumPy's implementation of matrix multiplication, and report the running time of each implementation. Please submit the running times reported by the `measure_time` function.

2 Word2Vec

In this part of the homework, you will apply your newfound expertise concerning Python matrix manipulation in the world of Word2Vec vectors. Please complete the following exercises by filling in the appropriate functions from `word2vec_exercise.py`.

2.1 Setup

Download the file `word2vec_embeddings.zip` from the Canvas site, which contains a set of 300-dimensional word vectors (i.e., each word is associated with a 300-dimensional vector of real numbers). This set of word vectors comes from work by Omer Levy and Yoav Goldberg, and it is the *Bag of Words ($k = 2$)* file listed on the following page:[5]

https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/

Expand the `.zip` archive and examine the format of the included `.txt` file. Note that this file is fairly large (390MB), and will be even larger (909MB) once it is expanded, so you should make sure that you have enough space on your computer's disk.

In order to allow you to work with the embeddings, we have provided the `load_embeddings` function, which will load the word embeddings from the `.txt` file into Python. The function will return a *vocabulary*—a list containing all the words that have embeddings—and a matrix where each row is a word embedding. The vocabulary and the word embedding

[5] It will be *much* faster for you to download the file from Canvas than from this link.

matrix are in the same order: the ith row of the word embedding matrix contains the embedding for the ith list item in the vocabulary.

2.2 Exercises

Problem 16. (6 Points) Complete the definition of the function `get_embedding`, which looks up the embedding of a given word from the vocabulary and the word embedding matrix.

Problem 17. (6 Points) Complete the definition of the function `cosine_sim` which computes the cosine similarity of a pair of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ using the formula we discussed in class:

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}.$$

Please use the function `np.linalg.norm` to compute $\|\boldsymbol{x}\|$ and $\|\boldsymbol{y}\|$.

Problem 18. (6 Points) Fill in the definition of the function `get_neighbors_of_embedding`. Given a word embedding $\boldsymbol{e}$, this function finds the top k words whose embeddings have the highest cosine similarity with $\boldsymbol{e}$. The top k words do not have to be in order.

Problem 19. (6 Points) Now use the functions you have defined to complete the definition of the function `get_neighbors_of_word`, which finds the top k words whose embeddings have the greatest cosine similarity to the embedding of a given word w, not including w. This function should operate in a manner similar to `get_neighbors_of_embedding`, except that its first argument should be a word from the vocabulary rather than a word embedding vector. Again, the top k words to not have to been in order.

Problem 20. (3 Points) What are the 5 most similar words to the following?

- cat
- mask
- tahiti
- september
- virus

Problem 21. (6 Points) Complete the definition of the function `analogy`, which performs the analogy task `w1:w2::w3:x` using vector arithmetic on the embeddings for the words `w1`, `w2` and `w3`:

$$[\![\texttt{x}]\!] = [\![\texttt{w2}]\!] - [\![\texttt{w1}]\!] + [\![\texttt{w3}]\!].$$

Your function should take `w1`, `w2`, and `w3` as input and return the k words whose embeddings have the greatest cosine similarity to $[\![\texttt{w2}]\!] - [\![\texttt{w1}]\!] + [\![\texttt{w3}]\!]$.

Problem 22. (4 Points) Test your analogy implementation on the following examples, looking both at the top answer and the 5 closest words:

- `dog:puppy::cat:x`
- `france:french::spain:x`
- `long:longest::fat:x`
- `give:gives::hope:x`
- `dog:puppy::cow:x`
- `france:french::afghanistan:x`
- `long:longest::white:x`
- `give:giving::hope:x`

Comment on the cases where it succeeds and those in which it does not.

3 Submission Instructions

You will submit your solutions to this assignment to CodePost, an online platform for grading assignments. Please create an account on CodePost at the following link: https://codepost.io/signup/join?code=PYKUYVYOU3. You must create your account using your `@yale.edu` email address and the invite code `PYKUYVYOU3`.

To submit your completed assignment, please upload the following files to CodePost. Please ensure that your files have the same filenames as indicated below.

- Your edited version of `numpy_exercise.py`, containing your solutions to Problems 9–14.
- Your edited version of `word2vec_exercise.py`, containing your solutions to Problems 16–22.
- A Markdown document called `assignment1.md`, containing your responses to Problems 1–8, 15, 20, and 22.