Homework 1

Problem 1 [5pts]:

Suppose there are 4 independent variables. And 3 operations, i.e., addition, multiplication, and power, are allowed in connecting them into an expression. How many different expression can we have? For example, let the 4 variabels be A, B , C, and D. Then A+B * C ^D is one expression while A^B * C+D is another. The order of variables matters, e.g., A+B and B+A are two different expressions. Each operator or variable appears in the expression EAXCTLY ONCE. No parentheses.

The purpose of this problem is to understand how many different parametric equations can exist among a set of variables, and thus using traditional scientific discovery way to model the relationship between high-dimensional variables is very challenging.

Problem 2 [5pts (2.5pts for correct returns, and 2.5pts for correct plot)]:

In the example code for Unit 1, there is a demo that the score of a neural network changes along with the maximal number of iterations (i.e., the `max_iter` argument in the function `test_NN`). Now, let's visualize the change.

Implement a function `learning_curve` with the following I/O specifications:

```python
def learning_curve(Ts, Hs, filename):

    # INSERT YOUR CODE HERE

    return max_iters, scores
```

where - the first two arguments `Ts` and `Hs` (1-D numpy array each, e.g., [1,2,3] not [[1],[2], [3]]) are the input and corresponding output for a supervised learning task, - the last argument `filename` (string) specifies the filename (Matplotlib uses the suffix to automatically determine the file format) to save the plot, - the 1st return `max_iters` (1-D numpy array) is a sequence of maximal numbers of iterations from 50 to 2000 with a step of 50, - and the 2nd return `scores` (1-D numpy array) is a sequence of scores returned from the function `test_NN`, each of which corresponds to an element in `max_iters`.

Make a line plot between the maximal number of iterations and the score of the NN, and save it as `filename`. Just do basic plot (`matplotlib.pyplot.plt(max_iters, scores)`) with all default settings. No labels nor title needed. Do not adjust figure size, resolution, tick values and locations, etc.

A file `hw1.py` is provided for you to jumpstart. It also includes to test cases. Just finish the definition of `learning_curve`. Do not change other non-commented lines. Then if you run `hw1.py`, you should expect to see the following output, if on Google CoLab:

```
<class 'numpy.ndarray'> [  50  100  150  200  250  300  350  400  450  500  550  600  650  7
  750  800  850  900  950 1000 1050 1100 1150 1200 1250 1300 1350 1400
 1450 1500 1550 1600 1650 1700 1750 1800 1850 1900 1950 2000]
<class 'numpy.ndarray'> [-24.51278849 -14.0049323   -7.28061896  -3.24700787  -1.02356304
   0.10299754   0.62719032   0.85084125   0.93854938   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779]
404eacc77aea113aa8ddad886283ed36

<class 'numpy.ndarray'> [  50  100  150  200  250  300  350  400  450  500  550  600  650  7
  750  800  850  900  950 1000 1050 1100 1150 1200 1250 1300 1350 1400
 1450 1500 1550 1600 1650 1700 1750 1800 1850 1900 1950 2000]
<class 'numpy.ndarray'> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
d66d8632c8afb82e0215e1d8d8418388
```

or this, if on PyRite.cs.iastate.edu:

```
<class 'numpy.ndarray'> [  50  100  150  200  250  300  350  400  450  500  550  600  650  7
  750  800  850  900  950 1000 1050 1100 1150 1200 1250 1300 1350 1400
 1450 1500 1550 1600 1650 1700 1750 1800 1850 1900 1950 2000]
<class 'numpy.ndarray'> [-24.51278849 -14.0049323   -7.28061896  -3.24700787  -1.02356304
   0.10299754   0.62719032   0.85084125   0.93854938   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779
   0.95801779   0.95801779   0.95801779   0.95801779   0.95801779]
0cc1ca4e5125acf90ec0031f245f7f97

<class 'numpy.ndarray'> [  50  100  150  200  250  300  350  400  450  500  550  600  650  7
  750  800  850  900  950 1000 1050 1100 1150 1200 1250 1300 1350 1400
 1450 1500 1550 1600 1650 1700 1750 1800 1850 1900 1950 2000]
<class 'numpy.ndarray'> [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
62f9834264b748f4424253b89c45add9
```

Note the iteration should go from 50 (including) to 2000 (including). Thus, 50, 100, 150, . . . , 1900, 1950, 2000.

How to submit

Just submit the modifield hw1.py file. For problem 1, strictly just one line of comment at the top. Just the number. Then insert lines to finish function definition to learning_curve below. Feel free to import numpy and matplotlib in your function definition. Do NOT import modules beyond numpy and matplotlib.