CS 540: Introduction to Artificial Intelligence
Homework Assignment # 9

Assigned: 4/16
Due: 4/23 before class

Hand in your homework:

If a homework has programming questions, please hand in the Java program. If a homework has written questions, please hand in a PDF file. Regardless, please zip all your files into hwX.zip where X is the homework number. Go to UW Canvas, choose your CS540 course, choose Assignment, click on Homework X: this is where you submit your zip file.

Late Policy:

All assignments are due at the beginning of class on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 9:30 a.m., and it is handed in between Wednesday 9:30 a.m. and Thursday 9:30 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

[60 points] Linear Regression - Body Weight vs Brain Weight

The dataset at `http://people.sc.fsu.edu/~jburkardt/datasets/regression/x01.txt` is a collection of average weight of the body and the weight of the brain for a number of mammal species. For this homework, we have already pre-processed the data in the file data.csv. Each line in this file corresponds to a mammal species with first column indicating the body weight and the second column indicating the brain weight. The values are separated by commas.

Write a program **BodyVsBrain.java** with the following command line format:

```
$java BodyVsBrain FLAG [arg1 arg2]
```

where the two optional arguments are real valued.

1. [5 *points*] When FLAG=100, we will compute the statistics of the dataset, like sample mean $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and sample standard deviation $\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2}$.

   Print n, the number of data points on the first line, the sample mean and the sample standard deviation of body weight on the second line, and the sample mean and the sample standard deviation of brain weight on the third line.

   For real values in this homework, keep four digits after decimal point (This is just for printing, do not round off the numbers for further calculations). For example (the numbers are made-up):

   ```
   $java BodyVsBrain 100
   100
   87.6543 40.1234
   32.1098 10.1234
   ```

2. [5 *points*] We will perform linear regression with the model

   $$f(x) = \beta_0 + \beta_1 x.$$

   We first define the mean squared error as a function of β_0, β_1:

   $$MSE(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i - y_i)^2.$$

   When FLAG=200, arg1=β_0 and arg2=β_1. Print the corresponding MSE.

   ```
   $java BodyVsBrain 200 0 0
   834962.5029
   $java BodyVsBrain 200 100.00 0
   805204.5061
   $java BodyVsBrain 200 -50.9 0.5
   251205.9923
   $java BodyVsBrain 200 50.55 50.55
   2299433922.4050
   ```

3. [10 *points*] We perform gradient descent on MSE. At current parameter (β_0, β_1), the gradient is defined by the vector of partial derivatives

$$\frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_0} = \frac{2}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i - y_i) \quad (1)$$

$$\frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_1} = \frac{2}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i - y_i)x_i. \quad (2)$$

When FLAG=300, arg1=β_0 and arg2=β_1. Print the corresponding gradient as two numbers on separate lines.

```
$java BodyVsBrain 300 0 0
-397.5800
-1650157.9772
$java BodyVsBrain 300 100.00 0
-197.5800
-1593531.1385
$java BodyVsBrain 300 -50.9 0.5
-216.2458
-747355.5257
$java BodyVsBrain 300 50.55 50.55
28328.3870
92565806.1970
```

4. [10 *points*] Gradient descent starts from initial parameter $(\beta_0^{(0)}, \beta_1^{(0)})$, and iterates the following updates at time $t = 1, 2, \ldots, T$:

$$\beta_0^{(t)} = \beta_0^{(t-1)} - \eta\frac{\partial MSE(\beta_0^{(t-1)}, \beta_1^{(t-1)})}{\partial \beta_0} \quad (3)$$

$$\beta_1^{(t)} = \beta_1^{(t-1)} - \eta\frac{\partial MSE(\beta_0^{(t-1)}, \beta_1^{(t-1)})}{\partial \beta_1}. \quad (4)$$

When FLAG=400, arg1=η and arg2=T. Start from initial parameter $(\beta_0^{(0)}, \beta_1^{(0)}) = (0, 0)$. Perform T iterations of gradient descent. Print the following in each iteration on a line, separated by space: $t, \beta_0^{(t)}, \beta_1^{(t)}, MSE(\beta_0^{(t)}, \beta_1^{(t)})$. For example,

```
$java BodyVsBrain 400 1e-7 5
1 0.0000 0.1650 588028.7174
2 0.0001 0.2993 424542.0532
3 0.0001 0.4085 316302.9585
4 0.0001 0.4974 244641.4458
5 0.0001 0.5698 197196.7363

$java BodyVsBrain 400 1e-8 5
```

```
1 0.0000 0.0165 807985.9714
2 0.0000 0.0327 782005.3556
3 0.0000 0.0486 756983.8884
4 0.0000 0.0642 732886.1600
5 0.0000 0.0795 709678.0680

$java BodyVsBrain 400 1e-9 5
1 0.0000 0.0017 832242.0182
2 0.0000 0.0033 829531.6620
3 0.0000 0.0049 826831.3965
4 0.0000 0.0066 824141.1842
5 0.0000 0.0082 821460.9876

$java BodyVsBrain 400 1e-5 5
1 0.0040 16.5016 227288400.6112
2 -0.0855 -274.4627 70632902935.6433
3 1.4727 4855.9687 21960124169280.0430
4 -26.0212 -85606.4253 6827522877409993.0000
5 458.7454 1509472.8203 2122714266189187072.0000
```

Note with $\eta = 10^{-5}$, gradient descent is diverging.

The following is not required for hand in, but try different initial parameters, η, and much larger T and see how small you can make MSE.

5. [10 *points*] Instead of using gradient descent, we can compute the closed-form solution for the parameters directly. For ordinary least squared in 1D, this is

$$\hat{\beta}_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

where $\bar{x} = (\sum x_i)/n$ and $\bar{y} = (\sum y_i)/n$. Required: When FLAG=500, print $\hat{\beta}_0, \hat{\beta}_1$ (note the order), and the corresponding MSE on a single line separated by space.

The following is not required for hand in, but give it some thought: what does a negative $\hat{\beta}_0$ mean?

6. [5 *points*] Using $\hat{\beta}_0, \hat{\beta}_1$ you can predict the brain weight of a species given the body weight. Required: When FLAG=600, arg1=body weight. Print a single real number which is the predicted brain weight for a species with that body weight. Use the closed-form solution to make the predictions.

   For example,

```
$java BodyVsBrain 600 500
394.6009
```

   The following is not required for hand in, but give it some thought:

- What's the prediction for the brain weight of a species with body weight 50?
- What does that say about the model?

7. [5 *points*] If you are agonizing over your inability to get gradient descent to match the closed-form solution in question 5, you are not alone. The culprit is the scale of input x compared to the scale of the implicit offset value 1 (think $\beta_0 = \beta_0 \cdot 1$). Gradient descent converges slowly when these scales differ greatly, a situation known as bad condition number in optimization. When FLAG=700, we ask you to first normalize the input x (note: not the labels y):

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5}$$

$$std_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{6}$$

$$x_i \leftarrow (x_i - \bar{x})/std_x. \tag{7}$$

Then proceed exactly as when FLAG=400. The two arguments are again arg1=η and arg2=T. For example,

```
$java BodyVsBrain 700 0.1 5
1 39.7580 165.2826 574430.4723
2 71.5644 298.0419 406416.1542
3 97.0095 404.6776 298065.1590
4 117.3656 490.3301 228190.2996
5 133.6505 559.1284 183128.3050

$java BodyVsBrain 700 1 5
1 397.5800 1652.8263 790900.6521
2 0.0000 53.3170 749635.6514
3 397.5800 1601.2292 710989.9691
4 0.0000 103.2496 674797.3426
5 397.5800 1552.9073 640902.0629

$java BodyVsBrain 700 0.01 5
1 3.9758 16.5283 806348.0413
2 7.8721 32.7313 778849.6074
3 11.6904 48.6155 752423.6728
4 15.4324 64.1871 727028.4073
5 19.0996 79.4523 702623.6118
```

With $\eta = 0.2$ your model should converge within 25 iterations.

(Note the β's are now for the normalized version of x, but you can easily translate them back for the original x with a little algebra. This is not required for the homework.)

8. [10 *points*] Now we implement Stochastic Gradient Descent (SGD). With everything the same as part 8 (including x normalization), we modify the definition of gradient in equations (1) and (2) as follows. In iteration t we randomly pick one of the n items. Say we picked (x_{j_t}, y_{j_t}). We approximate the gradient using that item only:

$$\frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_0} \approx 2(\beta_0 + \beta_1 x_{j_t} - y_{j_t}) \quad (8)$$

$$\frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_1} \approx 2(\beta_0 + \beta_1 x_{j_t} - y_{j_t}) x_{j_t}. \quad (9)$$

When FLAG=800, print the same information as in part 7. For example (your results will differ because of randomness in the items picked):

```
$java BodyVsBrain 800 0.1 5
1 0.3240 -0.0946 834990.2247
2 0.2655 -0.0769 834984.1173
3 2.2096 -0.4283 834796.9149
4 2.5849 -0.5191 834799.7060
5 107.9122 44.4145 732235.1437
```

Since out data set is small, there is little advantage of SGD over gradient descent. However, on large data sets SGD becomes more desirable.

9. Hints:
   1. Update β_0, β_1 simultaneously in an iteration. Don't use a new β_0 to calculate β_1.
   2. Use double instead of float in Java.
   3. Don't round the variables themselves to 4 digits in the middle stages. That is for printing only.