10/10

Homework 1 - Berkeley STAT 157

Handout 1/22/2017, due 1/29/2017 by 4pm in Git by committing to your repository. Please ensure that you add the TA Git account to your repository.

1. Write all code in the notebook.
2. Write all text in the notebook. You can use MathJax to insert math or generic Markdown to insert figures (it's unlikely you'll need the latter).
3. **Execute** the notebook and **save** the results.
4. To be safe, print the notebook as PDF and add it to the repository, too. Your repository should contain two files: `homework1.ipynb` and `homework1.pdf`.

The TA will return the corrected and annotated homework back to you via Git (please give `rythei` access to your repository).

In [1]:
```
from mxnet import ndarray as nd
from time import time
```

1. Speedtest for vectorization

Your goal is to measure the speed of linear algebra operations for different levels of vectorization. You need to use `wait_to_read()` on the output to ensure that the result is computed completely, since NDArray uses asynchronous computation. Please see http://beta.mxnet.io/api/ndarray/_autogen/mxnet.ndarray.NDArray.wait_to_read.html (http://beta.mxnet.io/api/ndarray/_autogen/mxnet.ndarray.NDArray.wait_to_read.html) for details.

1. Construct two matrices A and B with Gaussian random entries of size 4096×4096.
2. Compute $C = AB$ using matrix-matrix operations and report the time.
3. Compute $C = AB$, treating A as a matrix but computing the result for each column of B one at a time. Report the time.
4. Compute $C = AB$, treating A and B as collections of vectors. Report the time.
5. Bonus question - what changes if you execute this on a GPU?

In [2]:
```
import mxnet as mx
nd_a1 = mx.nd.array([[0, 0, 0]])
# nd_a1 = mx.nd.empty((0,3))
nd_a1 = mx.nd.concat(nd_a1, mx.nd.array([[1,2,3],[4,5,6]]), dim=0)
nd_a1 = mx.nd.concat(nd_a1, mx.nd.array([[7, 8, 9]]), dim=0)
print("\nnd_a1", nd_a1)
print(nd_a1.shape)
```

```
nd_a1
[[0. 0. 0.]
 [1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
<NDArray 4x3 @cpu(0)>
(4, 3)
```

In [3]:
```
A = nd.array([[1, 6],
              [2, 7]])
B = nd.array([[3, 4],
              [5, 8]])
print(nd.dot(A, B))
C = nd.dot(A, B[:, 0]).expand_dims(0)
print(C)
C = nd.concat(C, nd.array([[-1, -1]]), dim=0)
print(C)
D = nd.dot(A[0, :], B[:, 0]).expand_dims(0)
for i in range(2):
    for j in range(2):
        if i != 0 or j != 0:
            D = nd.concat(D, nd.dot(A[i, :], B[:, j]).expand_dims(0),
 dim=0)
D.reshape(2, 2)
```

```
[[33. 52.]
 [41. 64.]]
<NDArray 2x2 @cpu(0)>

[[33. 41.]]
<NDArray 1x2 @cpu(0)>

[[33. 41.]
 [-1. -1.]]
<NDArray 2x2 @cpu(0)>
```

Out[3]:
```
[[33. 52.]
 [41. 64.]]
<NDArray 2x2 @cpu(0)>
```

In [4]:
```
A = nd.random.normal(shape=(4096,4096))
B = nd.random.normal(shape=(4096,4096))
tic = time()
C = nd.dot(A, B)
C.wait_to_read()
print("Time for 1.2", time()-tic)
# Source https://stackoverflow.com/questions/49873467/how-to-append-a
-element-to-mxnet-ndarray
```

```
Time for 1.2 0.31929516792297363
```

In [5]:
```
tic = time()
C = nd.dot(A, B[:, 0]).expand_dims(0)
for i in range(1, 4096):
    C = nd.concat(C, nd.dot(A, B[:, i]).expand_dims(0), dim=0)
C = C.T
print(C.shape)
C.wait_to_read()
print("Time for 1.3", time()-tic)
```

```
(4096, 4096)
Time for 1.3 27.757354259490967
```

In [6]:
```
tic = time()
C = nd.empty((4096, 4096))
for i in range(4096):
    for j in range(4096):
        C[i, j] = nd.dot(A[i, :], B[:, j])
C.wait_to_read()
print("Time for 1.4", time()-tic)
```

```
Time for 1.4 6639.831993341446
```

2. Semidefinite Matrices

Assume that $A \in \mathbb{R}^{m \times n}$ is an arbitrary matrix and that $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix with nonnegative entries.

1. Prove that $B = ADA^\top$ is a positive semidefinite matrix.
2. When would it be useful to work with B and when is it better to use A and D?

2.1) To show

$$ADA^T$$

is a positive semidefinite (psd) matrix, we must show that

$$x^T ADA^T x >= 0 \ \forall x \epsilon R^n$$

$$x^T ADA^T x = (A^T x)^T D(A^T x)$$

Substitue $z = A^T x$ we get

$$z^T Dz$$

D is psd, since the sum of it's diagonal entries is equal to the sum of its eigenvalues and is nonnegative, so $z^T Dz >= 0$ for all z, so ADA^T is positive semidefinite. QED

2.2) It is useful to work with B since it is a square, positiive semidefinite matrix. It seems to hold all the information of A but transform an arbitrary matrix into one with nice properties. When we do SVD we look at AA^T, so now we can specify which entries to leave out by setting some of D's diagonal values to zero or which entries to modify by setting D's diagonal values to not 1.

If say A is a data matrix, it is better to work with it directly when training a machine learning algorithm.

3. MXNet on GPUs

1. Install GPU drivers (if needed)
2. Install MXNet on a GPU instance
3. Display `!nvidia-smi`
4. Create a 2×2 matrix on the GPU and print it. See http://d2l.ai/chapter_deep-learning-computation/use-gpu.html (http://d2l.ai/chapter_deep-learning-computation/use-gpu.html) for details.

In [1]:
```
!nvidia-smi
```

```
Tue Jan 29 09:58:35 2019
+-----------------------------------------------------------------------------
---------+
| NVIDIA-SMI 396.44                 Driver Version: 396.44
        |
|-------------------------------+----------------------+----------------------
---------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Unc
orr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Co
mpute M. |
|===============================+======================+======================
=========|
|   0  Tesla P100-PCIE...  Off  | 00000000:04:00.0 Off |
       0 |
| N/A   29C    P0    31W / 250W |      0MiB / 16280MiB |      0%
Default |
+-------------------------------+----------------------+----------------------
---------+
|   1  Tesla P100-PCIE...  Off  | 00000000:05:00.0 Off |
       0 |
| N/A   31C    P0    29W / 250W |     10MiB / 16280MiB |      0%
Default |
+-------------------------------+----------------------+----------------------
---------+
|   2  Tesla P100-PCIE...  Off  | 00000000:08:00.0 Off |
       0 |
| N/A   35C    P0    32W / 250W |   9049MiB / 16280MiB |      0%
Default |
+-------------------------------+----------------------+----------------------
---------+
|   3  Tesla P100-PCIE...  Off  | 00000000:09:00.0 Off |
       0 |
| N/A   31C    P0    30W / 250W |    379MiB / 16280MiB |      0%
Default |
+-------------------------------+----------------------+----------------------
---------+
|   4  Tesla P100-PCIE...  Off  | 00000000:84:00.0 Off |
       0 |
| N/A   33C    P0    32W / 250W |      0MiB / 16280MiB |      0%
Default |
+-------------------------------+----------------------+----------------------
---------+
|   5  Tesla P100-PCIE...  Off  | 00000000:85:00.0 Off |
       0 |
| N/A   33C    P0    31W / 250W |      0MiB / 16280MiB |      0%
Default |
+-------------------------------+----------------------+----------------------
---------+
|   6  Tesla P100-PCIE...  Off  | 00000000:88:00.0 Off |
       0 |
| N/A   30C    P0    30W / 250W |      0MiB / 16280MiB |      0%
Default |
+-------------------------------+----------------------+----------------------
---------+
|   7  Tesla P100-PCIE...  Off  | 00000000:89:00.0 Off |
       0 |
```

```
| N/A   34C    P0    29W / 250W |      0MiB / 16280MiB |      4%
Default |
+-------------------------------+----------------------+------------
---------+


+-----------------------------------------------------------------------
---------+
| Processes:                                                            GP
U Memory |
|  GPU       PID   Type   Process name                                  Us
age      |
|=======================================================================
=========|
|    2     30451      C   /usr/bin/python3
9037MiB |
|    3      2932      C   ...wmzhang/anaconda3/envs/cs189/bin/python
  369MiB |
+-----------------------------------------------------------------------
---------+
```

In [8]:
```
nd.random.normal(shape=(2,2), ctx=mx.gpu())
```

Out[8]:
```
[[-1.3204551   0.68232244]
 [-0.9858383   0.01992839]]
<NDArray 2x2 @gpu(0)>
```

4. NDArray and NumPy

Your goal is to measure the speed penalty between MXNet Gluon and Python when converting data between both. We are going to do this as follows:

1. Create two Gaussian random matrices A, B of size 4096×4096 in NDArray.
2. Compute a vector $\mathbf{c} \in \mathbb{R}^{4096}$ where $c_i = ||AB_{i.}||^2$ where $\mathbf{c}$ is a **NumPy** vector.

To see the difference in speed due to Python perform the following two experiments and measure the time:

1. Compute $||AB_{i.}||^2$ one at a time and assign its outcome to $\mathbf{c}_i$ directly.
2. Use an intermediate storage vector $\mathbf{d}$ in NDArray for assignments and copy to NumPy at the end.

In [9]:
```
import numpy as np
A = nd.random.normal(shape=(4096, 4096), ctx=mx.gpu())
B = nd.random.normal(shape=(4096, 4096), ctx=mx.gpu())
c = np.zeros(4096)
tic = time()
for i in range(4096):
    c[i] = (nd.dot(A, B[:, i]).norm() ** 2).asscalar()
print("Time for 4.1", time()-tic)
```

```
Time for 4.1 3.652972936630249
```

In [10]:
```
d = nd.zeros(4096, ctx=mx.gpu())
tic = time()
for i in range(4096):
    d[i] = (nd.dot(A, B[:, i]).norm() ** 2)
d = nd.dot(A, B).norm() ** 2
c = d.asnumpy()
print("Time for 4.1", time()-tic)
```

```
Time for 4.1 1.4525656700134277
```

5. Memory efficient computation

We want to compute $C \leftarrow A \cdot B + C$, where A, B and C are all matrices. Implement this in the most memory efficient manner. Pay attention to the following two things:

1. Do not allocate new memory for the new value of C.
2. Do not allocate new memory for intermediate results if possible.

In [11]:
```
C = nd.empty((4096, 4096), ctx=mx.gpu())
C += nd.dot(A, B)
```

6. Broadcast Operations

In order to perform polynomial fitting we want to compute a design matrix A with

$$A_{ij} = x_i^j$$

Our goal is to implement this **without a single for loop** entirely using vectorization and broadcast. Here $1 \leq j \leq 20$ and $x = \{-10, -9.9, \dots 10\}$. Implement code that generates such a matrix.

In [12]:
```
x = nd.arange(-100, 101)/10
A = x.reshape((200, 1)).broadcast_to((200, 20)) ** \
nd.arange(1, 21).broadcast_to((200, 20))
A
```

Out[12]:
```
[[-1.0000000e+01  1.0000000e+02 -1.0000000e+03 ...  9.9999998e+17
  -1.0000000e+19  1.0000000e+20]
 [-9.8999996e+00  9.8009995e+01 -9.7029889e+02 ...  8.3451318e+17
  -8.2616803e+18  8.1790629e+19]
 [-9.8000002e+00  9.6040001e+01 -9.4119208e+02 ...  6.9513558e+17
  -6.8123289e+18  6.6760824e+19]
 ...
 [ 9.6999998e+00  9.4089996e+01  9.1267297e+02 ...  5.7795107e+17
   5.6061250e+18  5.4379413e+19]
 [ 9.8000002e+00  9.6040001e+01  9.4119208e+02 ...  6.9513558e+17
   6.8123289e+18  6.6760824e+19]
 [ 9.8999996e+00  9.8009995e+01  9.7029889e+02 ...  8.3451318e+17
   8.2616803e+18  8.1790629e+19]]
<NDArray 200x20 @cpu(0)>
```