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 [43]:
```
from mxnet import ndarray as nd
import numpy as np
```

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 [7]:
```
import time
tic = time.time()
a = nd.random.normal(0, 1, (4096, 4096))
b = nd.random.normal(0, 1, (4096, 4096))
c = nd.dot(a, b)
print(time.time() - tic)
c.wait_to_read()
print(time.time() - tic)
```

```
0.010117053985595703
3.6122050285339355
```

In [7]:
```
tic = time.time()
b_t = b.T
c = nd.zeros((4096, 4096))
for i in range(4096):
    c[i] = nd.dot(a, b_t[i])
c = c.T
print(time.time() - tic)
c.wait_to_read()
print(time.time() - tic)
```

```
4.0372560024261475
73.63805103302002
```

In [44]:
```
tic = time.time()
b_t = b.T
c = nd.zeros((4096, 4096))
for i in range(4096):
    for j in range(4096):
        c[j, i] = nd.sum(a[j] * b_t[i])
c.wait_to_read()
print(time.time() - tic)
```

```
4665.045080900192
```

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?

saved as pdf

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.

Tried to run GPU, got up to gpu access on AWS but had trouble connecting it with jupyter notebook

In [15]:
```
!nvidia-smi
print(nd.zeros((2,2)))
```

```
/bin/sh: nvidia-smi: command not found

[[0. 0.]
 [0. 0.]]
<NDArray 2x2 @cpu(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 $c_i = \|AB_{i\cdot}\|^2$ 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 $\|AB_{i\cdot}\|^2$ 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 [42]:
```
tic = time.time()
a = nd.random.normal(0, 1, (4096, 4096))
b = nd.random.normal(0, 1, (4096, 4096))
c = np.zeros(4096)
b_t = b.T
for i in range(4096):
    vec = nd.dot(a, b_t[i])
    c[i] = vec.norm().asscalar()
print(time.time() - tic)
```

```
72.5136399269104
```

In [41]:
```
tic = time.time()
a = nd.random.normal(0, 1, (4096, 4096))
b = nd.random.normal(0, 1, (4096, 4096))
c = nd.zeros(4096)
b_t = b.T
print(time.time() - tic)
for i in range(4096):
    vec = nd.dot(a, b_t[i])
    c[i] = vec.norm()
c = c.asnumpy()
print(time.time() - tic)
```

```
0.051258087158203125
66.14804792404175
```

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 [45]:
```
a = nd.random.normal(0, 1, (4096, 4096))
b = nd.random.normal(0, 1, (4096, 4096))
c = nd.random.normal(0, 1, (4096, 4096))
nd.elemwise_add(nd.dot(a, b), c, out = c)
c
```

Out[45]:
```
[[-136.53722     72.59831     -2.1823547 ...  -89.08128     42.181435
   -16.420313 ]
 [ -97.03846      6.6538477   70.85523   ...  -51.745754   -44.10528
    20.53577  ]
 [  16.992313   -10.352718    42.555477  ...  -53.673546    49.54098
   -23.488247 ]
 ...
 [ -73.42694    124.763466    47.24751   ...   12.638767   -30.496918
    56.542774 ]
 [ 130.45993     23.206944     1.0797606 ...  -74.54492      5.094515
    48.01982  ]
 [ -49.40635     27.739756    68.296074  ...    9.620451   -68.24221
    65.92578  ]]
<NDArray 4096x4096 @cpu(0)>
```

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 $x = \{-10, -9.9, \ldots 10\}$. Implement code that generates such a matrix.

In [40]:
```
x = nd.arange(-10, 10, .1).reshape((200,1))
j = nd.arange(1, 21).reshape((1, 20))
nd.broadcast_power(x, j)
```

Out[40]:
```
[[-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]
 ...
 [ 9.7000008e+00  9.4090012e+01  9.1267322e+02 ...  5.7795210e+17
   5.6061355e+18  5.4379519e+19]
 [ 9.8000011e+00  9.6040024e+01  9.4119232e+02 ...  6.9513681e+17
   6.8123415e+18  6.6760952e+19]
 [ 9.8999996e+00  9.8009995e+01  9.7029889e+02 ...  8.3451318e+17
   8.2616803e+18  8.1790629e+19]]
<NDArray 200x20 @cpu(0)>
```

Window Help

/derek-tang-b59634139/

My Proxy Server EZproxy UCB Dell CS 189/289A: Intr...

* Required

Desktop — ubuntu@ip-172-31-29-104: ~ — -bash

~ — -bash | jupyter-n...k • python | .../local/cuda — -bash | ubuntu@ip-172-31-...

```
Length: 414428 (405K) [application/octet-stream]
Saving to: 'cuda_9.1.128_mac_network'

cuda_9.1.128_mac_ne 100%[===================>] 404.71K  --.-KB/s    in 0.05s

2019-01-29 22:26:54 (7.31 MB/s) - 'cuda_9.1.128_mac_network' saved [414428/414428]

ubuntu@ip-172-31-29-104:~$ sudo sh cuda_9.1.128_mac_network
cuda_9.1.128_mac_network: 1: cuda_9.1.128_mac_network: x??1au???@GD??Fd?r: not found
cuda_9.1.128_mac_network: 1: cuda_9.1.128_mac_network: #9?Pa??
                                                        0: not found
cuda_9.1.128_mac_network: 1: cuda_9.1.128_mac_network: ??m?66??:?0??6??@#@: not found
cuda_9.1.128_mac_network: 5: cuda_9.1.128_mac_network: Syntax error: "(" unexpected
ubuntu@ip-172-31-29-104:~$ echo fixing cuda
fixing cuda
ubuntu@ip-172-31-29-104:~$ which cuda
ubuntu@ip-172-31-29-104:~$ sudo rm /usr/local/cuda
rm: cannot remove '/usr/local/cuda': No such file or directory
ubuntu@ip-172-31-29-104:~$ ls
cuda_9.1.128_mac  cuda_9.1.128_mac_network
ubuntu@ip-172-31-29-104:~$ cd ..
ubuntu@ip-172-31-29-104:/home$ ls
ubuntu
ubuntu@ip-172-31-29-104:/home$ cd ubuntu
ubuntu@ip-172-31-29-104:~$ packet_write_wait: Connection to 52.36.78.169 port 22: Broken pi
calvisitor-10-105-195-146:~ Derek$ ssh -i "keypair.pem" ubuntu@ec2-52-36-78-169.us-west-2.c
ssh: connect to host ec2-52-36-78-169.us-west-2.compute.amazonaws.com port 22: Operation ti
```

Undergraduate student at UC Berkeley

combinepdf (21).pdf new doc 2019-....pdf derek_tang_res....pdf

2 a) $B = ADA^T$ if ADA^T is semipos, so is $A^T D A$

if $x^T(A^T D A^T)x \geq 0$ then it is semi definite positive

~~$(x^T A) D (A^T x)$~~ y

$\Rightarrow (x^T A^T) D (A x)$ $\qquad \underline{y = Ax}$

$\Rightarrow y^T D y$ $\rightarrow$ since D is semi positive

& $y^T y \geq 0$ (inner product

$x^T (A^T D A) x \geq 0$ & $A^T D A$ is semipositive

b) we use B if m is smaller than n significantly since B is a $m \times m$ matrix, if n is smaller than we want to work with 2 $m \times n$, $n \times n$ matrices A & D as this will be more memory efficient.