9/10

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 [2]:
```
from mxnet import ndarray as nd
import time
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 [5]:
```
# 1. Construct two matrices A and B with Gaussian random entries of size
  4096×4096.
A = nd.random.normal(0, 1, shape=(4096, 4096))
B = nd.random.normal(0, 1, shape=(4096, 4096))
print("A is ", A)
print("B is ", B)
# 2. Compute C=AB using matrix-matrix operations and report the time.
tic = time.time()
C = nd.dot(A,B)
C.wait_to_read()
print("C is ", C)
print("Q2 takes", time.time()- tic)

# 3. Compute C=AB, treating A as a matrix but computing the result for e
ach column of B one at a time. Report the time.
tic = time.time()
C=nd.empty((4096,4096))
for i in range(4096):
    C[:,i] = nd.dot(A,B[:,i])
C.wait_to_read()
print("C is ", C)
print("Q3 takes", time.time() - tic)

# 4. Compute C=AB, treating A and B as collections of vectors. Report th
e time.
tic = time.time()
C=nd.zeros((4096,4096))
for i in range(4096):
    a = A[:,i].reshape(4096,1)
    b = B[i,:].reshape(1,4096)
    C += nd.dot(a,b)
C.wait_to_read()
print("C is ", C)
print("Q4 takes", time.time() - tic)
```

−1 this just get diagonals

want for i=1,… 4096
for j=1,…4096
C_{ij}=dot(A[i,:],B[:,j])

```
A is
[[ 0.04010138 -0.82130146  0.02005161 ...  0.80797803  0.0349124
  -0.9425538 ]
 [ 0.4579651   0.35110465 -2.7191465  ...  0.2302415   0.2839324
  -0.54985106]
 [-1.1534044   0.30501708  0.80644757 ... -1.6631485   0.69836503
   0.6364544 ]
 ...
 [ 0.5691745  -1.0284108  -0.515387   ...  0.40762436  1.068956
  -0.01077725]
 [ 0.34715003 -0.70612836 -0.7524027  ... -0.4507234   1.2779481
   0.852945  ]
 [ 0.24677981 -0.17408593  1.8073856  ...  0.7218124  -0.4130202
  -0.07617839]]
<NDArray 4096x4096 @cpu(0)>
B is
[[ 1.2563426e+00  2.0983241e+00 -6.1626399e-01 ...  7.1981925e-01
  -1.5064095e-01 -1.6847348e+00]
 [-5.9105557e-01 -7.4600556e-04  3.9198685e-01 ... -2.2434056e+00
   2.0302789e+00  6.1633265e-01]
 [-6.1852258e-01  2.1420236e-01 -5.6821045e-02 ...  4.1352261e-02
  -1.0406296e+00  1.0598879e+00]
 ...
 [ 2.7899129e+00  1.4155358e-01  1.0943099e+00 ...  5.1718676e-01
  -1.2406477e+00 -8.1241745e-01]
 [-1.1760486e+00  2.6735411e+00 -2.4220882e-01 ... -3.0564606e-01
   1.5179449e+00  2.4704537e+00]
 [-1.3527721e+00  7.0983845e-01 -2.0442135e+00 ... -1.5420042e+00
  -1.7140442e-01  4.6152738e-03]]
<NDArray 4096x4096 @cpu(0)>
C is
[[ -13.285672   -28.294012  -120.399765  ...  112.58602    -35.496662
  -109.732544 ]
 [  30.453777    23.37611    -56.675674  ...   24.243011    49.816593
    15.3279705]
 [ -24.976767   -35.074642   -30.914444  ...   -4.885561   -10.728184
    76.08944  ]
 ...
 [-151.3319     107.07407   -125.48469   ...   56.779675   129.2976
  -109.78065  ]
 [ -45.615746   -22.664043    -7.4296336 ...  121.454994   -58.20906
   -95.325264 ]
 [ -82.03247     -5.521243    30.89802   ...  -11.5960655 -122.28372
     1.2381835]]
<NDArray 4096x4096 @cpu(0)>
Q2 takes 1.3674461841583252
C is
[[ -13.285627   -28.294062  -120.39978   ...  112.58604    -35.496643
  -109.732544 ]
 [  30.453712    23.376108   -56.67568   ...   24.242981    49.81656
    15.32799  ]
 [ -24.976753   -35.07466    -30.914482  ...   -4.8855286  -10.728184
    76.08942  ]
 ...
 [-151.33191    107.074066  -125.484665  ...   56.779675   129.29756
  -109.78069  ]
 [ -45.615738   -22.664051    -7.4296474 ...  121.45499    -58.209015
```

```
  -95.32531  ]
 [ -82.03244    -5.521265    30.89798   ...  -11.596052  -122.28375
     1.2382298]]
<NDArray 4096x4096 @cpu(0)>
Q3 takes 23.638182163238525
C is
[[ -13.2856455  -28.294075  -120.39991   ...  112.58585    -35.4967
  -109.73244  ]
 [  30.453823    23.376108   -56.675602  ...   24.242912    49.816586
    15.327974 ]
 [ -24.976778   -35.074673   -30.914282  ...   -4.885559   -10.728128
    76.08936  ]
 ...
 [-151.3321     107.074196  -125.484535  ...   56.77968    129.2975
  -109.780495 ]
 [ -45.615623   -22.664017    -7.4296207 ...  121.45502    -58.209126
   -95.32538  ]
 [ -82.03248    -5.521231    30.89808   ...  -11.596074  -122.283905
     1.2382019]]
<NDArray 4096x4096 @cpu(0)>
Q4 takes 179.12236976623535
```

1. Bonus question - what changes if you execute this on a GPU?

C is [[-13.2856455 -28.294075 -120.39991 ... 112.58585 -35.4967 -109.73244] [30.453823 23.376108 -56.675602 ... 24.242912 49.816586 15.327974] [-24.976778 -35.074673 -30.914282 ... -4.885559 -10.728128 76.08936] ... [-151.3321 107.074196 -125.484535 ... 56.77968 129.2975 -109.780495] [-45.615623 -22.664017 -7.4296207 ... 121.45502 -58.209126 -95.32538] [-82.03248 -5.521231 30.89808 ... -11.596074 -122.283905 1.2382019]] Q5 takes 1.7138192653656006

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?

✓

1. Let x be an arbitrary non-zero vector. $x^T B x = x^T A D A^T x = (A^T x)^T D A^T x = b^T D b = \sum_{i=1}^{n} \lambda_i b_i^2 >= 0$. so B is a PSD
2. B is diagnolizable. Assume the eigenvalues are postive, then we can get a basis of orthonormal eigenvectors, many calculations become easier using orthonormal eigenbasis. When calculating the power of B, it's easy to use ADA^T

→ can only get powers like this when A is orthogonal

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.

ubuntu@ip-172-31-37-85:~$ nvidia-smi Sun Jan 27 08:13:31 2019 +---+ | NVIDIA-SMI 410.79 Driver Version: 410.79 CUDA Version: 10.0 | |-------------------------------+----------------------+----------------------+ | GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC | | Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. | |===============================+======================+======================| | 0 Tesla V100-SXM2... On | 00000000:00:1E.0 Off | 0 | | N/A 36C P0 26W / 300W | 0MiB / 16130MiB | 0% Default | +-------------------------------+----------------------+----------------------+ +---+ | Processes: GPU Memory | | GPU PID Type Process name Usage | |===| | No running processes found | +---+ [[0. 1.] [2. 3.]]

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 [4]:
```
A = nd.random.normal(shape=(4096,4096))
B = nd.random.normal(shape=(4096,4096))
# 1. Compute ‖ABi‖2 one at a time and assign its outcome to ci directly.
c = np.zeros((4096,1))
start = time.time()
for i in range(4096):
    c[i] = nd.norm(nd.dot(A, B[:,i])).asscalar()
end = time.time()
print("Method 1 takes ", end - start)

# 2. Use an intermediate storage d in NDArray for assignments and copy t
o NumPy at the end.

d = nd.zeros((4096,1))
start = time.time()
for i in range(4096):
    d[i] = nd.norm(nd.dot(A, B[:,i])).asscalar()
c = d.asnumpy()
end = time.time()
print("Method 2 takes ", end - start)
```

```
Method 1 takes  25.370314121246338
Method 2 takes  23.677567958831787
```

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 [4]:
```
C = nd.zeros((4096,4096))
#tic = time.time()
C = nd.elemwise_add(nd.dot(A,B),C)
#print(time.time()-tic)
#tic = time.time()
nd.elemwise_add(nd.dot(A,B),C,out=C)
#print(time.time()-tic)
```

Out[4]:
```
[[-6.43394279e+00 -1.37317566e+02 -8.94156723e+01 ...  6.33354111e+01
   7.78073883e+01  8.15101318e+01]
 [-1.49630295e+02  1.58471893e+02  6.12498245e+01 ... -5.57203674e+00
   1.27422363e+02 -9.79201736e+01]
 [-1.26177185e+02 -1.60241714e+02  5.05774918e+01 ...  3.18762283e+01
  -1.06060280e+02  2.81837463e-01]
 ...
 [ 1.89660625e+01 -1.08990974e+01  1.35774994e+02 ...  1.20331802e+02
  -3.09089050e+01  1.33221344e+02]
 [ 9.73447418e+01  2.68981476e+01 -1.48852234e+02 ... -2.40328922e+01
  -1.02886139e+02  1.24706306e+02]
 [-1.58903694e+00 -2.09539986e+01 -3.99479187e+02 ... -3.73300819e+01
   7.10207214e+01  1.08717667e+02]]
<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 [8]:
```
j=nd.arange(1,21)
x=nd.arange(-10,11).reshape(21,1)
nd.broadcast_power(x,j)
#x**j
```

Out[8]:
```
[[-1.00000000e+01  1.00000000e+02 -1.00000000e+03  1.00000000e+04
  -1.00000000e+05  1.00000000e+06 -1.00000000e+07  1.00000000e+08
  -1.00000000e+09  1.00000000e+10 -9.99999980e+10  9.99999996e+11
  -9.99999983e+12  1.00000000e+14 -9.99999987e+14  1.00000003e+16
  -9.99999984e+16  9.99999984e+17 -9.99999998e+18  1.00000002e+20]
 [-9.00000000e+00  8.10000000e+01 -7.29000000e+02  6.56100000e+03
  -5.90490000e+04  5.31441000e+05 -4.78296900e+06  4.30467200e+07
  -3.87420480e+08  3.48678451e+09 -3.13810596e+10  2.82429522e+11
  -2.54186593e+12  2.28767931e+13 -2.05891136e+14  1.85302015e+15
  -1.66771819e+16  1.50094642e+17 -1.35085174e+18  1.21576651e+19]
 [-8.00000000e+00  6.40000000e+01 -5.12000000e+02  4.09600000e+03
  -3.27680000e+04  2.62144000e+05 -2.09715200e+06  1.67772160e+07
  -1.34217728e+08  1.07374182e+09 -8.58993459e+09  6.87194767e+10
  -5.49755814e+11  4.39804651e+12 -3.51843721e+13  2.81474977e+14
  -2.25179981e+15  1.80143985e+16 -1.44115188e+17  1.15292150e+18]
 [-7.00000000e+00  4.90000000e+01 -3.43000000e+02  2.40100000e+03
  -1.68070000e+04  1.17649000e+05 -8.23543000e+05  5.76480100e+06
  -4.03536080e+07  2.82475264e+08 -1.97732672e+09  1.38412872e+10
  -9.68890122e+10  6.78223086e+11 -4.74756153e+12  3.32329302e+13
  -2.32630511e+14  1.62841363e+15 -1.13988947e+16  7.97922632e+16]
 [-6.00000000e+00  3.60000000e+01 -2.16000000e+02  1.29600000e+03
  -7.77600000e+03  4.66560000e+04 -2.79936000e+05  1.67961600e+06
  -1.00776960e+07  6.04661760e+07 -3.62797056e+08  2.17678234e+09
  -1.30606940e+10  7.83641641e+10 -4.70184985e+11  2.82110984e+12
  -1.69266591e+13  1.01559954e+14 -6.09359759e+14  3.65615856e+15]
 [-5.00000000e+00  2.50000000e+01 -1.25000000e+02  6.25000000e+02
  -3.12500000e+03  1.56250000e+04 -7.81250000e+04  3.90625000e+05
  -1.95312500e+06  9.76562500e+06 -4.88281240e+07  2.44140624e+08
  -1.22070310e+09  6.10351565e+09 -3.05175777e+10  1.52587895e+11
  -7.62939441e+11  3.81469721e+12 -1.90734863e+13  9.53674336e+13]
 [-4.00000000e+00  1.60000000e+01 -6.40000000e+01  2.56000000e+02
  -1.02400000e+03  4.09600000e+03 -1.63840000e+04  6.55360000e+04
  -2.62144000e+05  1.04857600e+06 -4.19430400e+06  1.67772160e+07
  -6.71088640e+07  2.68435456e+08 -1.07374182e+09  4.29496730e+09
  -1.71798692e+10  6.87194767e+10 -2.74877907e+11  1.09951163e+12]
 [-3.00000000e+00  9.00000000e+00 -2.70000000e+01  8.10000000e+01
  -2.43000000e+02  7.29000000e+02 -2.18700000e+03  6.56100000e+03
  -1.96830000e+04  5.90490000e+04 -1.77147000e+05  5.31441000e+05
  -1.59432300e+06  4.78296900e+06 -1.43489070e+07  4.30467200e+07
  -1.29140160e+08  3.87420480e+08 -1.16226150e+09  3.48678451e+09]
 [-2.00000000e+00  4.00000000e+00 -8.00000000e+00  1.60000000e+01
  -3.20000000e+01  6.40000000e+01 -1.28000000e+02  2.56000000e+02
  -5.12000000e+02  1.02400000e+03 -2.04800000e+03  4.09600000e+03
  -8.19200000e+03  1.63840000e+04 -3.27680000e+04  6.55360000e+04
  -1.31072000e+05  2.62144000e+05 -5.24288000e+05  1.04857600e+06]
 [-1.00000000e+00  1.00000000e+00 -1.00000000e+00  1.00000000e+00
  -1.00000000e+00  1.00000000e+00 -1.00000000e+00  1.00000000e+00
  -1.00000000e+00  1.00000000e+00 -1.00000000e+00  1.00000000e+00
  -1.00000000e+00  1.00000000e+00 -1.00000000e+00  1.00000000e+00
  -1.00000000e+00  1.00000000e+00 -1.00000000e+00  1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
   1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
```

```
   1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
   1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
   1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00]
 [ 2.00000000e+00  4.00000000e+00  8.00000000e+00  1.60000000e+01
   3.20000000e+01  6.40000000e+01  1.28000000e+02  2.56000000e+02
   5.12000000e+02  1.02400000e+03  2.04800000e+03  4.09600000e+03
   8.19200000e+03  1.63840000e+04  3.27680000e+04  6.55360000e+04
   1.31072000e+05  2.62144000e+05  5.24288000e+05  1.04857600e+06]
 [ 3.00000000e+00  9.00000000e+00  2.70000000e+01  8.10000000e+01
   2.43000000e+02  7.29000000e+02  2.18700000e+03  6.56100000e+03
   1.96830000e+04  5.90490000e+04  1.77147000e+05  5.31441000e+05
   1.59432300e+06  4.78296900e+06  1.43489070e+07  4.30467200e+07
   1.29140160e+08  3.87420480e+08  1.16226150e+09  3.48678451e+09]
 [ 4.00000000e+00  1.60000000e+01  6.40000000e+01  2.56000000e+02
   1.02400000e+03  4.09600000e+03  1.63840000e+04  6.55360000e+04
   2.62144000e+05  1.04857600e+06  4.19430400e+06  1.67772160e+07
   6.71088640e+07  2.68435456e+08  1.07374182e+09  4.29496730e+09
   1.71798692e+10  6.87194767e+10  2.74877907e+11  1.09951163e+12]
 [ 5.00000000e+00  2.50000000e+01  1.25000000e+02  6.25000000e+02
   3.12500000e+03  1.56250000e+04  7.81250000e+04  3.90625000e+05
   1.95312500e+06  9.76562500e+06  4.88281240e+07  2.44140624e+08
   1.22070310e+09  6.10351565e+09  3.05175777e+10  1.52587895e+11
   7.62939441e+11  3.81469721e+12  1.90734863e+13  9.53674336e+13]
 [ 6.00000000e+00  3.60000000e+01  2.16000000e+02  1.29600000e+03
   7.77600000e+03  4.66560000e+04  2.79936000e+05  1.67961600e+06
   1.00776960e+07  6.04661760e+07  3.62797056e+08  2.17678234e+09
   1.30606940e+10  7.83641641e+10  4.70184985e+11  2.82110984e+12
   1.69266591e+13  1.01559954e+14  6.09359759e+14  3.65615856e+15]
 [ 7.00000000e+00  4.90000000e+01  3.43000000e+02  2.40100000e+03
   1.68070000e+04  1.17649000e+05  8.23543000e+05  5.76480100e+06
   4.03536080e+07  2.82475264e+08  1.97732672e+09  1.38412872e+10
   9.68890122e+10  6.78223086e+11  4.74756153e+12  3.32329302e+13
   2.32630511e+14  1.62841363e+15  1.13988947e+16  7.97922632e+16]
 [ 8.00000000e+00  6.40000000e+01  5.12000000e+02  4.09600000e+03
   3.27680000e+04  2.62144000e+05  2.09715200e+06  1.67772160e+07
   1.34217728e+08  1.07374182e+09  8.58993459e+09  6.87194767e+10
   5.49755814e+11  4.39804651e+12  3.51843721e+13  2.81474977e+14
   2.25179981e+15  1.80143985e+16  1.44115188e+17  1.15292150e+18]
 [ 9.00000000e+00  8.10000000e+01  7.29000000e+02  6.56100000e+03
   5.90490000e+04  5.31441000e+05  4.78296900e+06  4.30467200e+07
   3.87420480e+08  3.48678451e+09  3.13810596e+10  2.82429522e+11
   2.54186593e+12  2.28767931e+13  2.05891136e+14  1.85302015e+15
   1.66771819e+16  1.50094642e+17  1.35085174e+18  1.21576651e+19]
 [ 1.00000000e+01  1.00000000e+02  1.00000000e+03  1.00000000e+04
   1.00000000e+05  1.00000000e+06  1.00000000e+07  1.00000000e+08
   1.00000000e+09  1.00000000e+10  9.99999980e+10  9.99999996e+11
   9.99999983e+12  1.00000000e+14  9.99999987e+14  1.00000003e+16
   9.99999984e+16  9.99999984e+17  9.99999998e+18  1.00000002e+20]]
<NDArray 21x20 @cpu(0)>
```

```
In [ ]:
```