HW1

January 28, 2019

```
In [1]: import mxnet as mx
        from mxnet import nd
        from mxnet.gluon import nn
        import numpy as np

        import time
```

0.1 1.1

0.1.1 1

```
In [98]: A = nd.random.normal(0, 1, shape=(4096, 4096))
         B = nd.random.normal(0, 1, shape=(4096, 4096))
```

2

```
In [99]: tic = time.time()

         C = nd.dot(A, B)
         C.wait_to_read()

         print(time.time() - tic)
```

```
0.4232213497161865
```

3

```
In [18]: tic = time.time()

         for i in range(4096):
             if i == 0:
                 C = nd.dot(A, B[:, :1])
             else:
                 C = nd.concat(C, nd.dot(A, B[:, i:i+1]))

                 C.wait_to_read()

         print(time.time() - tic)
```

```
129.4343912601471
```

4

```
In [19]: tic = time.time()

         for i in range(4096):
             for j in range(4096):
                 if j == 0:
                     row = nd.dot(A[i:i+1, :], B[:, :1])
                 else:
                     row = nd.concat(row, nd.dot(A[i:i+1, :], B[:, j:j+1]))
             if i == 0:
                 C = row
             else:
                 C = nd.concat(C, row, dim=0)

         C.wait_to_read()

         print(time.time() - tic)
```

```
1747.2818703651428
```

5 For a small data which can be fit into CPU without overflowing, CPU will be faster. However, if the data doesn't fit into the buffers and is big enough, using GPU will be faster as GPU can store more data and compute asynchronous.

```
In [6]: A = nd.random.normal(0, 1, shape=(4096, 4096), ctx=mx.gpu(0))
        B = nd.random.normal(0, 1, shape=(4096, 4096), ctx=mx.gpu(0))

        tic = time.time()

        for i in range(4096):
            for j in range(4096):
                if j == 0:
                    row = nd.dot(A[i:i+1, :], B[:, :1])
                else:
                    row = nd.concat(row, nd.dot(A[i:i+1, :], B[:, j:j+1]))
            if i == 0:
                C = row
            else:
                C = nd.concat(C, row, dim=0)

        C.wait_to_read()

        print(time.time() - tic)
```

```
1669.3835248947144
```

0.1.2 1.2

1 To prove $B = ADA^T$ is PSD, we need to check if all the diagonal entries are equal or greater than 0 or that $ADA^T \geq 0$. We can rewrite them in vector form as $\sum_i a_i d_i a_i^T \geq 0$. This again becomes $\sum_i a_i^2 d_i$ and since each vector of d is nonnegative and for any real value of a, $a^2 \geq 0$, the sum will be at least 0 or greater, proving that B or ADA^T is a positive semidefinite.

2 We know that the matrix B is symmetric and that ADA^T is the eigendecomposition of it. By the properties of eigenvectors of a symetric matrix, we know that those are orthogonal to each other. Hence we prefer to use ADA^T when we would like to get the matrix to the power of any real number. For example, it is better to compute $(ADA^T)^2$ which just becomes AD^2A^T than B^2. Since D is a diagonal matrix, the square matrix or any powers of it is easier to compute than doing B^n as it computes some redundant procedures. Other than that, it would be better to use B.

0.1.3 1.3

In [20]: `!nvidia-smi`

```
Mon Jan 28 20:05:14 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.48                 Driver Version: 410.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 1080    Off  | 00000000:65:00.0  On |                  N/A |
| 26%   32C    P8    11W / 180W |    696MiB /  8116MiB |      9%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1243      G   /usr/lib/xorg/Xorg                            18MiB |
|    0      1296      G   /usr/bin/gnome-shell                          49MiB |
|    0      1566      G   /usr/lib/xorg/Xorg                           284MiB |
|    0      1697      G   /usr/bin/gnome-shell                         140MiB |
|    0      2064      G   ...quest-channel-token=8467226019251121638   201MiB |
+-----------------------------------------------------------------------------+
```

```
In [21]: D = nd.random.normal(0, 1, shape=(2, 2), ctx=mx.gpu(0))
         D
```

```
Out[21]:
         [[-1.3204551   0.68232244]
          [-0.9858383   0.01992839]]
         <NDArray 2x2 @gpu(0)>
```

0.1.4 1.4

```
In [3]: A = nd.random.normal(0, 1, shape=(4096, 4096))
        B = nd.random.normal(0, 1, shape=(4096, 4096))
```

Computing directly to c_i

```
In [4]: tic = time.time()

        c = np.zeros(4096)

        for i in range(4096):

            c[i] = (nd.dot(A, B[:, i]).norm()**2).asscalar()

        print(time.time() - tic)
```

```
15.849517583847046
```

With intermediate storage of NDArray

```
In [5]: tic = time.time()

        d = nd.empty(4096)

        for i in range(4096):

            nd.norm(nd.dot(A, B[:, i]), out=d[i])
            d[i] = d[i]**2

        d.wait_to_read()
        d = d.asnumpy()
        print(time.time() - tic)
```

```
13.085696935653687
```

0.1.5 1.5

1

In [36]:
```
A = nd.arange(12).reshape((3,3))
B = nd.arange(12).reshape((3,3))
C = nd.arange(12).reshape((3,3))

C[:] = nd.dot(A, B) + C

C
```

Out[36]:
```
[[ 15.   19.   23.]
 [ 45.   58.   71.]
 [ 75.   97. 119.]]
<NDArray 3x3 @cpu(0)>
```

2

In [55]:
```
A = nd.arange(12).reshape((3,3))
B = nd.arange(12).reshape((3,3))
C = nd.arange(12).reshape((3,3))

nd.elemwise_add(nd.dot(A, B), C, out=C)
```

Out[55]:
```
[[ 15.   19.   23.]
 [ 45.   58.   71.]
 [ 75.   97. 119.]]
<NDArray 3x3 @cpu(0)>
```

0.1.6 1.6

In [24]:
```
x = nd.array(np.around(np.arange(-10, 10.1, 0.1), decimals=1)).reshape((-1,1))
y = nd.arange(1, 21).reshape((1, -1))

A = x**y

A
```

Out[24]:
```
[[-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]
```

```
 ...
 [ 9.8000002e+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]
 [ 1.0000000e+01  1.0000000e+02  1.0000000e+03 ...  9.9999998e+17
   1.0000000e+19  1.0000000e+20]]
<NDArray 201x20 @cpu(0)>
```