In [13]:
```
import numpy as np
```

Let's create some fake data

In [14]:
```
n = 65000 # Num examples
k = 64 # Num cluster means
d = 12 # Dim of data
```

In [15]:
```
# Let's generate a random data matrix:
X = np.random.randn(n,d)

# Let's assume we have cluster means like so:
ClusterMeans = np.random.randn(k,d)
```

Pure Python: compute distance^2 between all data pts and all means?

- We can try this using pure python code
- Hint: this is going to be slower
- The output of these functions will be an $n \times k$ matrix D
- Where $D_{i,j} = \|x_i - \mu_j\|^2$

In [16]:
```
def compute_distances_in_pure_python(X,ClusterMeans):
    return np.array(
        [
            [
                np.linalg.norm(X[i,:] - ClusterMeans[j,:])**2
                for j in range(k)
            ] for i in range(n)
        ])
```

In [17]:
```
# The ipython notebook has a cool way to time operations
%time compute_distances_in_pure_python(X,ClusterMeans)
```

```
CPU times: user 25.3 s, sys: 194 ms, total: 25.5 s
Wall time: 25.5 s
```

Out[17]:
```
array([[ 15.35237989,  39.0002268 ,  33.6241462 , ...,  15.47464467,
         17.27953859,  15.12896553],
       [ 18.66645437,  32.12109167,  23.35419891, ...,  19.17769189,
         32.5398458 ,  15.05740036],
       [ 14.61649942,  27.35928937,  24.39918403, ...,  16.84951338,
         20.15390182,  13.50165235],
       ...,
       [ 24.26760505,  24.41454467,  19.90878896, ...,  21.22972889,
         42.34108537,  27.94083864],
       [ 11.20988906,  21.16240199,  23.09072738, ...,  34.96384391,
         48.79927704,  25.95277422],
       [ 20.48260675,  32.4313865 ,  32.37389488, ...,  23.05791543,
         27.94394063,  14.10456026]])
```

Numpy Version: compute distance^2 between all data pts and all means?

- A big hint: numpy does matrix operations **super** quickly
- Let M be the matrix of cluster means, i.e. where row j is μ_j
- It so happens that the matrix D (as defined above) can be written in terms of the following matrix operations

$$D = X^2_{\text{rows}} \mathbf{1}_k^\top + \mathbf{1}_n M^{2\top}_{\text{rows}} - 2XM^\top$$

- Here I'm using X^2_{rows} and M^2_{rows} are the square of norms of rows of X and M
- I'm using $\mathbf{1}_k$ and $\mathbf{1}_n$ as a vector of k and n 1's.

In [18]:
```
def compute_distances_using_numpy(X,ClusterMeans):
    X_row_norms = np.linalg.norm(X,axis=1) ** 2
    M_row_norms = np.linalg.norm(ClusterMeans,axis=1) ** 2
    D = (np.outer(X_row_norms, np.ones(k))
         + np.outer(np.ones(n), M_row_norms)
         - 2 * np.dot(X,ClusterMeans.T))
    return D
```

In [19]:
```
%time compute_distances_using_numpy(X,ClusterMeans)
```

```
CPU times: user 112 ms, sys: 78.1 ms, total: 190 ms
Wall time: 151 ms
```

Out[19]:
```
array([[ 15.35237989,  39.0002268 ,  33.6241462 , ...,  15.47464467,
         17.27953859,  15.12896553],
       [ 18.66645437,  32.12109167,  23.35419891, ...,  19.17769189,
         32.5398458 ,  15.05740036],
       [ 14.61649942,  27.35928937,  24.39918403, ...,  16.84951338,
         20.15390182,  13.50165235],
       ...,
       [ 24.26760505,  24.41454467,  19.90878896, ...,  21.22972889,
         42.34108537,  27.94083864],
       [ 11.20988906,  21.16240199,  23.09072738, ...,  34.96384391,
         48.79927704,  25.95277422],
       [ 20.48260675,  32.4313865 ,  32.37389488, ...,  23.05791543,
         27.94394063,  14.10456026]])
```

In [ ]: