Gradient Descent: Time Complexity

- Recall: Time complexity for matrix multiplication
    - [A x B] * [B x C] is O(ABC). If A=B=C, then is $O(A^3)$
- Assume we do **k** iterations
- Our code is:
    - pred = x @ cur_weights
    - error = pred - t
    - gradient = x.T @ error / examples
- Pred: [nxd] * [dx1] ⇒ O(nd) and output is [nx1]
- Error: O(n), just subtracting two 1D arrays
- Gradient: [dxn] * [nx1] ⇒ O(nd)
- As we do our algorithm max k iterations, then complexity is **O(knd)**

Normal Equations: Time Complexity

- X is nxd matrix (n examples, each is d features)
- $\Theta = (X^T X)^{-1} X^T y$
- $X^T X$: is [dxn] * [nxd] = $O(d^2 n)$ and output is H = dxd matrix
- H^{-1}= is $O(d^3)$ for inverting
- $X^T y$: [dxn] * [nx1] = $O(dn)$ and output is R = dx1 matrix
- H^{-1} x R = [dxd] * [dx1] = $O(d^2)$
  - Observe, it is less operations to divide the operations (H/R) than doing sequential multiplications
  - As otherwise we do H^{-1} x X^T which is [dxd] * [dxn] ⇒ $O(d^2 n)$
  - In other words: ABC matrices can be grouped: ((AxB)xC) or (A(BxC))
- In total: $O(d^2 n) + O(dn) + O(d^2)$ = **$O(d^2 n)$**

Comparison

- **O(k** x nd**) vs O(d** x nd**)**
- ~3000 iterations (k) is usually sufficient (if not too much) for many datasets
- If d is huge (e.g. 50k), gradient descent will be way faster

Side note for Matrix Form for derivatives

$$\text{MSE, } J(\underline{\theta}) = \frac{1}{m} \sum_j (y^{(j)} - \hat{y}(x^{(j)}))^2$$

$$= \frac{1}{m} \sum_j (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T})^2$$

$$\underline{\theta} = [\theta_0, \ldots, \theta_n]$$

$$\underline{y} = \left[y^{(1)} \ldots, y^{(m)} \right]^T$$

$$\underline{X} = \begin{bmatrix} x_0^{(1)} & \ldots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \ldots & x_n^{(m)} \end{bmatrix}$$

$$J(\underline{\theta}) = \frac{1}{m} (\underline{y}^T - \underline{\theta}\, \underline{X}^T) \cdot (\underline{y}^T - \underline{\theta}\, \underline{X}^T)^T$$

```
# Python / NumPy:
e = Y – X.dot( theta.T );
J = e.T.dot( e ) / m  # = np.mean( e ** 2 )
```