

PHW1

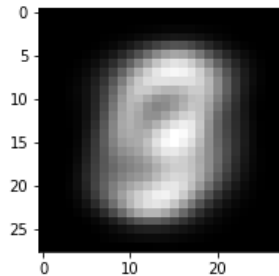
姓名: 吳添毅

系所: 網媒所

學號: R10944011

Part1: PCA

Q1.

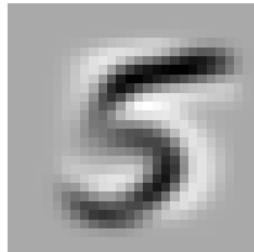
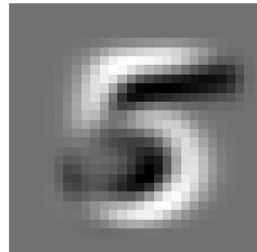


Q2.

$\lambda = 515302.09314403037$

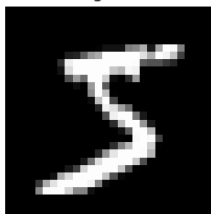
$\lambda = 296723.79901744524$

$\lambda = 217327.96293556504$



Q3.

Original 5



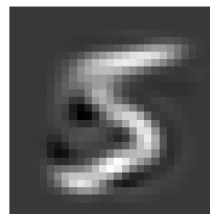
5 with 3d



5 with 10d



5 with 30d

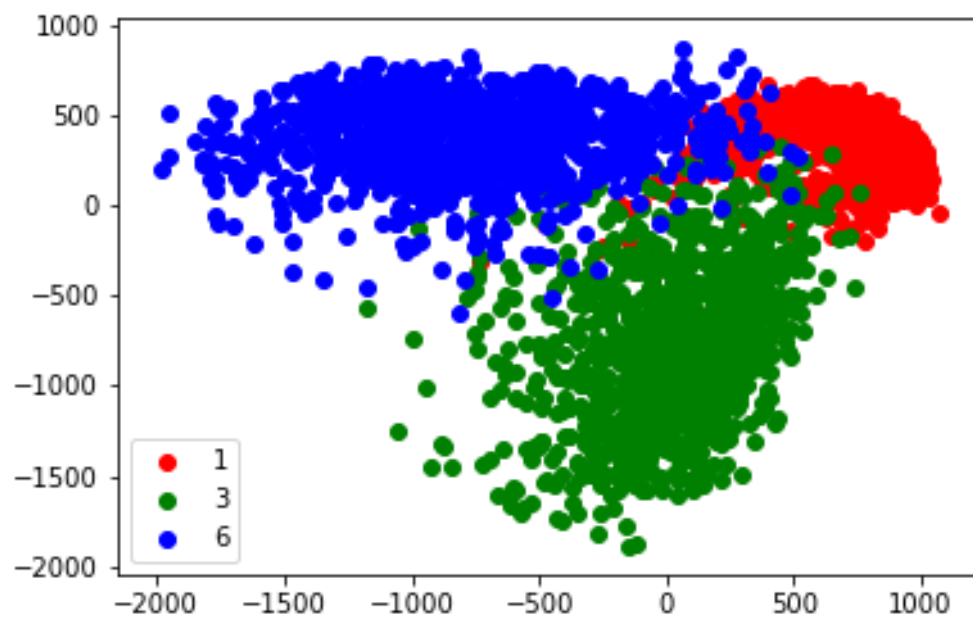


5 with 100d



使用 3、10 個 **bases** 所重建出來的 5 形狀跟原本的 5 還有蠻大的落差，但從 30 個 **bases** 後重建出來的結果基本上就和原圖的輪廓蠻像的了，到 100 個 **bases** 重建的結果基本跟原圖一樣，差了一些亮暗的地方而已。

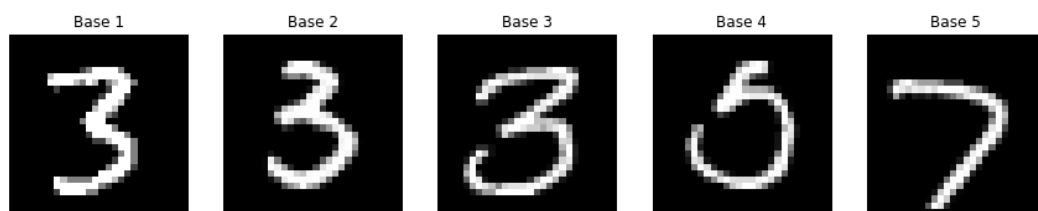
Q4.



可以發現經過 PCA 降維投影到 2 維平面上的結果發現不同數字會各自形成一個 cluster。

Part2: OMP

Q5.



可以發現 Base1, 2, 3 都剛好是 3 這個數字，可以得知說其實 Base 的圖若是原本就跟原圖 "3" 很接近的話就越有可能當 base。

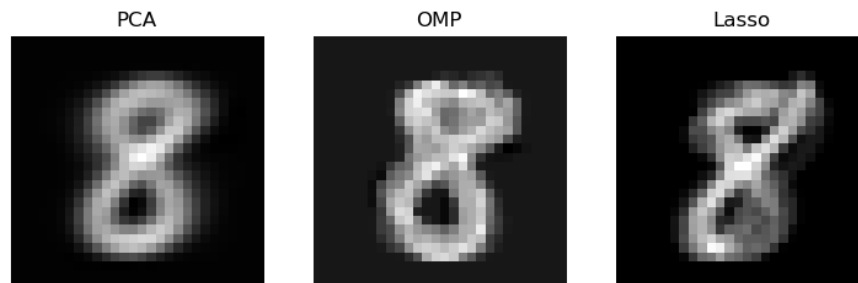
Q6.



Sparsity 為 5 的時候雖然已經有 8 的形狀出現了，但還有很多雜訊以及背景不夠黑，但可以看到若是增加 sparsity(不為 0) 數值的話，可以發現雜訊會慢慢減少。

Part3.

Q7.



上圖為 1. 2. 3. 結果。其中:

PCA 使用 5 個 bases (L2 norm=1263.8858...)

OMP sparsity 為 5 (L2 norm=905.2425...)

Lasso 使用預設參數且使用 coefficients 最高的 5 個 bases (L2 norm=1744.7328...)

4. 其實調整參數並且 plot 出來其實肉眼看不出什麼差距，頂多原本會出現 ConvergenceWarning，max_iter 調高一點就不會出現了。L2 norm 出現 3.7795...(alpha=0.15, max_iter=10000)，而且跑很久，alpha 值越高代表 penalty 越高，越難收斂，alpha=1 時最難收斂，max_iter=50000 還無法收斂(L2 norm=18.9240...)，可以發現如果沒收斂 L2 norm 還是會比收斂的時候大一些。

Bonus:

1. 定義好一些符號: (格式為 “程式內變數”: “對應公式的符號”)

N: N (number of features: 784)

p: p (number of bases: 6824)

x: x (bases with size: $N * p$ and is pre-normalized to mean 0 & variance 1)

B: β (coefficient with size p)

y: y (original signal)

alpha: λ (penalty)

r_j: r_{ij} (partial residuals)

2. 定義一些 function 方便計算:

計算 soft-thresholding: $S_a(x) = \text{sign}(x)\max(|x| - a, 0)$

```
# Soft-threshold:  $S_a(x) = \text{sign}(x)\max(|x| - a, 0)$ 
def soft_threshold(B, alpha):
    if B > 0:
        return max(abs(B) - alpha, 0)
    else:
        return -max(abs(B) - alpha, 0)
```

計算 partial residuals:

依照此公式 $r_{ij} = y_i - \sum_{k \neq j} x_{ik} \beta_k$ 直接計算 r_j 的 residual

```
def compute_partial_residual(j, y, x, B):
    assert B.shape[0] == x.shape[1], "Size of x is not equal to size of B"
    B[j] = 0
    sum_of_product_exclude_j = np.dot(x, B)
    r_j = y - sum_of_product_exclude_j
    return r_j
```

計算 Simple least square coefficient:

依照公式 $\beta_j^* = \frac{1}{N} \sum_{i=1}^N x_{ij} r_{ij}$ 算出 β_j^*

```
def least_square_coefficient(j, x, r, N):
    x_j = x[:, j]
    return (np.dot(x_j, r)) / N
```

3. 主程式架構

初始化 $B(6824 \times 1)$ 為 0。接著用跑規定的 iteration 次數，每個 iteration 都針對 6824 個 basis 做下列事情:

- a. 計算 partial residuals
- b. 計算 β_j^*
- c. 更新 β_j

```
def lasso(x, y, alpha=0.5, max_iter=500):
    N, p = x.shape
    B = np.zeros(p)
    for iter in range(max_iter):
        for j in range(p):
            r_j = compute_partial_residual(j, y, x, B)
            B_star = least_square_coefficient(j, x, r_j, N)
            B[j] = soft_threshold(B_star, alpha) / (x[:, j]**2).sum()
    return B
```

結果圖(參數 $\alpha = 0.5$, $\max_iter=2$):

L2-norm: 2298.6070499682282

Lasso HandCraft

