Elementos de Cálculo Numérico / Cálculo Numérico

Primer Cuatrimestre 2021

Octavo ejercicio computacional
31-05-21 al Lunes 07-06-21
Recuerde subir el archivo en formato ejercicioX_NOMBREAPELLIDO.py
Recuerde enviar su código al hacer consultas

En este ejercicio exploraremos una de las aplicaciones del método de descomposición en valores singulares (SVD). Este consiste en resumir información, reduciendo la cantidad de espacio en la memoria para guardar un objeto a cambio de pérdida de información. Hemos visto que podemos descomponer $A \in \mathbb{R}^{m\times n}$ en tres matrices $U \in \mathbb{R}^{m\times m}$, $S \in R^{m\times n}$ y $V \in \mathbb{R}^{n\times n}$, de forma tal de poder rearmar $A = USV^T$. Como ya sabemos que S es diagonal, en realidad podemos almacenar sólo su diagonal, y las dimensiones pertinentes, ahorrando espacio. En esta lógica, la compresión se realiza reduciendo la cantidad de elementos de la diagonal que preservamos de S. La idea es que a medida que incorporamos más elementos de la diagonal, más información mantendremos de la imagen. Reducir el tamaño de S ahorra memoria porque nos permite reducir la cantidad de elementos de U y V que necesitamos guardar (al haber tantos ceros en S, muchos valores no serán relevantes).

Mucho más modestamente, el objetivo de este ejercicio es explorar como cambia la calidad de la compresión a medida que incorporamos más valores singulares a la imagen.

Para esto:

A Emplearemos la función `svd` de modulo de álgebra lineal de numpy:

```
import numpy.linalg as npl
npl.svd()
```

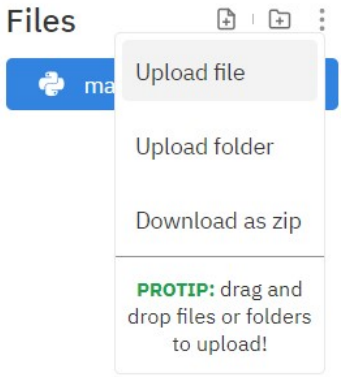
Esta función retorna tres arrays, `U,s,V`, (donde `V` es V^T y `s` es un vector con la diagonal de S) de forma tal que podemos recuperar la matriz `A` de la siguiente forma:

```
A = np.array([[1,2,3],[4,5,6]])
U,s,V = npl.svd(A)
S = np.zeros((A.shape[0], A.shape[1]))
S[:len(s), :len(s)] = np.diag(s)
A_ = np.dot(U,np.dot(S,V))
print(np.mean(A_-A))
```

No esperen que de cero (error de punto flotante), pero el error es muy pequeño. Exploren los tamaños de cada una de estas tres matrices, usando el comando `X.shape`, donde `X` es la matriz.

Nota: En general, para una dada dimension de un array `x[a:b:c]` tomará los elementos de `x` entre `a` y `b-1`, espaciando cada `c` pasos. Si queremos ir hasta el final del array, podemos omitir b, y si queremos mantener todos los elementos, sencillamente ponemos un `:`.

IMPORTANTE: en este trabajo utilizaremos, a modo de prueba, cinco archivos de imágenes, que están adjuntos a esta consigna. Descargar esos archivos y moverlos a la carpeta donde se encuentre el archivo `.py` de esta entrega. Si utiliza replit, los archivos de las imágenes pueden subirse haciendo clic en los tres puntos, arriba a la izquierda, y luego en *upload file*.



B Cargue y grafique la imagen `arbol.jpg` provista:

```
import imageio as img
image = img.imread('arbol.jpg',format='jpg')
```

Esto importará la imagen como una matriz de $m \times n \times 3$, donde esa tercer dimensión corresponde con los colores de la matriz. Construya una función que retorne las imágenes correspondientes a cada color. Aproveche el siguiente modelo:

```
def descompone_imagen(image):
    R = image[:,:,0] # Matriz de rojos
    G = image[:,:,1] # Matriz de verdes
    B = image[:,:,2] # Matriz de azules
    return(#COMPLETAR)
```

Compare el resultado de su función cada imagen resultante. Para graficar, si `A` es un array:

```
plt.imshow(A,cmap='gray',vmin=0,vmax=255)
plt.savefig('imagen.jpg')
```

El comando `cmap` sirve para indicar el mapa de colores (en este caso graficamos cada color en grises), y `vmax,vmin` indican el rango de valores para el máximo y mínimo de intensidad.

C Pruebe sobre la matriz de verdes la descomposición en valores singulares y compare ambas imagenes (la imagen en verdes original, y la recuperada usando la descomposición). ¿Es satisfactorio el resultado?

```
R, G, B = descompone_imagen(image)
U, s, V = #COMPLETAR
S = np.zeros((G.shape[0], G.shape[1]))
S[:len(s), :len(s)] = np.diag(s)
G_ = #COMPLETAR

plt.clf()
plt.imshow(G_, cmap='gray', vmin=0, vmax=255)
plt.savefig('imagen_recompuesta.png')
```

D Construya una función que tome la matriz `A`, aplique SVD, convierta en cero el último `p` porcentaje de elementos de `s`, y devuelva una matriz `A_`, resultante de recomponer la matriz con sólo `1-p` porcentaje de los valores singulares:

```
def reduce_svd(A,p):
    U,s,V = ## COMPLETAR ##
    n_elementos = int(p*len(##COMPLETAR))
    s[#COMPLETAR#] = 0
    S = np.zeros((A.shape[0], A.shape[1]))
    S[:len(s), :len(s)] = np.diag(s)
    A_ = ## COMPLETAR ##
    return(A_)
```

Pruebe el resultado de aplicar esta función a la imagen en verdes de `arbol.jpg` manteniendo un 10% de los valores singulares de la matriz (es decir, convirtiendo en cero un 90% de los elementos de `s`).

```
G_reducida = reduce_svd(#COMPLETAR, #COMPLETAR)
plt.clf()
plt.imshow(G_reducida, cmap='gray', vmin=0, vmax=255)
plt.savefig('imagen_gris_reducida.png')
```

E Ahora construya una función que calcule la descomposición SVD para cada una de las tres componentes de la imagen, y la reconstruya:

```
def comprime_svd(imagen,p):
    R,G,B = #COMPLETAR
    R_ = reduce_svd(#COMPLETAR)
    G_ = # COMPLETAR
    B_ = # COMPLETAR
    imagen_comprimida = np.zeros((R_.shape[0],R_.shape[1],3))
    imagen_comprimida[:,:,0] = # COMPLETAR
    imagen_comprimida[:,:,1] = # COMPLETAR
    imagen_comprimida[:,:,2] = # COMPLETAR
    return(imagen_comprimida)
```

F Implemente una función que calcule el error promedio en la imagen en función del porcentaje de valores singulares que elimina:

```
def errores_de_compresion(image):
    error = []
    p_values = np.arange(0, 1, .02)
    for p in ## COMPLETAR ##:
     print('Calculando con p='+str(p))
        image_ = comprime_svd(image,p)
        error.append(np.mean(np.abs(image_-image))/np.mean(image))
    return np.array(error)
```

¿Cuantos valores singulares considera necesarios para aproximar la imagen? Compare su criterio visualmente (graficando algunas imágenes) con lo que observa del gráfico.

G Empleando las otras imágenes provistas, grafique todas las curvas de error (`p` vs. error) en una misma imagen y compare. ¿Cambia el resultado con la complejidad de la imagen?

```
plt.figure()
p_values = np.arange(0, 1, .02)
imagenes = ['arbol.jpg', 'mona_lisa.jpg', 'fractal.jpg', 'poligono.jpeg', 'cuadrado
for imagePath in imagenes:
    print(f'Trabajando en imagen {imagePath}')
    image = img.imread(imagePath)
    errores = #COMPLETAR
    plt.plot(#COMPLETAR, #COMPLETAR, label=imagePath)
plt.legend()
plt.savefig('errores.png')
```

Nota: La resolución de este item puede demorar.