Q1

June 17, 2020

[1]:
```python
import numpy as np
import tensorly as tl
from tensorly.decomposition import tucker

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
<IPython.core.display.HTML object>
```

Question 1. Tensor Decomposition Reconstructions (15 points)

Part 1. Kruskal tensors are a way of representing tensor decompositions as a weighted sum of outer products.

$\chi = \sum_r \lambda_r U_{1r} \circ U_{2r} \circ \ldots \circ U_{nr}$ for each rank of the decomposition, r, and rank of the original tensor, n.

a) Given the following rank-2 CP decomposition:

$$\lambda = (39.288\, 10.676)\, U_1 = \begin{pmatrix} 0.5719 & 0.1469 \\ 0.5885 & 0.9817 \\ 0.5715 & -0.1210 \end{pmatrix} U_2 = \begin{pmatrix} 0.5121 & -0.4042 \\ 0.6284 & 0.5877 \\ 0.5856 & 0.7009 \end{pmatrix}$$

$$U_3 = \begin{pmatrix} 0.5605 & -0.3179 \\ 0.4921 & -0.3682 \\ 0.6661 & 0.8737 \end{pmatrix} U_4 = \begin{pmatrix} 0.7502 & -0.9201 \\ 0.6612 & 0.3917 \end{pmatrix}$$

Write out the calculation of the first outer product $U_{1,1} \circ U_{2,1}$

Answer:

$U_{1,1} \circ U_{2,1} = \mathrm{U}_{1,1} * \mathrm{U}_{2,1}{}^{\mathrm{T}}$

Where $\mathrm{U}_{1,1} = \begin{pmatrix} 0.5719 \\ 0.5885 \\ 0.5715 \end{pmatrix}$ and $\mathrm{U}_{2,1}{}^{\mathrm{T}} = \begin{pmatrix} 0.5121 & 0.6284 & 0.5856 \end{pmatrix}$

$$\begin{pmatrix} (0.5719 * 0.5121) & (0.5719 * 0.6284) & (0.5719 * 0.5856) \\ (0.5885 * 0.5121) & (0.5885 * 0.6284) & (0.5885 * 0.5856) \\ (0.5715 * 0.5121) & (0.5715 * 0.6284) & (0.5715 * 0.5856) \end{pmatrix} = \boxed{\begin{pmatrix} 0.2929 & 0.3594 & 0.3349 \\ 0.3014 & 0.3698 & 0.3446 \\ 0.2927 & 0.3591 & 0.3347 \end{pmatrix}}$$

0.0.1 Confirm written solution programmatically

[2]:
```
u11 = np.array([0.5719, 0.5885, 0.5715])
u21 = np.array([0.5121, 0.6284, 0.5856])
np.outer(u11, u21)
```

[2]:
```
array([[0.29286999, 0.35938196, 0.33490464],
       [0.30137085, 0.3698134 , 0.3446256 ],
       [0.29266515, 0.3591306 , 0.3346704 ]])
```

0.1 b) Either by hand or in code, calculate:

0.1.1 $\lambda_1 \; U_{1,1} \circ U_{2,1} \circ U_{3,1} \circ U_{4,1}$

[3]:
```
lam = np.array([39.288 , 10.676])

u1 = np.array([[0.5719, 0.1469], [0.5885,0.9817], [0.5715, -0.1210]])

u2 = np.array([[0.5121, -0.4042], [0.6284,0.5877], [0.5856, 0.7009]])

u3 = np.array([[0.5605, -0.3179], [0.4921,-0.3682], [0.6661, 0.8737]])

u4 = np.array([[0.7502, -0.9201], [0.6612,0.3917]])
```

[4]:
```
#intialize empty placeholder for outer product of the first rank
prod1 = np.zeros((3,3,3,2))

#extract first column from each U matrix
u11 = u1[:,0]
u21 = u2[:,0]
```

```
u31 = u3[:,0]
u41 = u4[:,0]

#perform outer product
for i in range(len(u41)):             #length should be 2 since there's 2␣
 ↪values in the columns of U4
    for j in range(len(u31)):         #length should be 3 since there's 3␣
 ↪values in the columns of U3
        for k in range(len(u21)):     #length should be 3 since there's 3␣
 ↪values in the columns of U2
            for l in range(len(u11)): #length should be 3 since there's 3␣
 ↪values in the columns of U1

                prod1[l][k][j][i] = u11[l] * u21[k] * u31[j] * u41[i]

#multiply outer product by the first lambda value
prod1 = prod1 * lam[0]
prod1
```

[4]:
```
array([[[[4.8382407 , 4.26425586],
         [4.24781132, 3.7438721 ],
         [5.74978078, 5.06765536]],

        [[5.93702491, 5.23268577],
         [5.21250661, 4.59412073],
         [7.05557946, 6.21854058]],

        [[5.5326572 , 4.87629024],
         [4.85748547, 4.28121754],
         [6.57502758, 5.79499899]]],


       [[[4.97867573, 4.38803038],
         [4.37110852, 3.85254193],
         [5.91667423, 5.2147494 ]],

        [[6.10935331, 5.38456999],
         [5.36380511, 4.72746992],
         [7.26037509, 6.39904027]],

        [[5.6932484 , 5.01782971],
         [4.99847911, 4.40548439],
         [6.76587469, 5.96320494]]],


       [[[4.83485672, 4.26127335],
         [4.24484031, 3.74125355],
```

```
          [5.74575925, 5.06411093]],

         [[5.93287241, 5.22902591],
          [5.20886087, 4.5909075 ],
          [7.05064463, 6.21419119]],

         [[5.52878753, 4.87287965],
          [4.85408804, 4.27822315],
          [6.57042886, 5.79094583]]]])
```

0.1.2 $\lambda_1\ U_{1,2} \circ U_{2,2} \circ U_{3,2} \circ U_{4,2}$

[5]:

```
#initialize empty placeholder for outer product fo the second rank
prod2 = np.zeros((3,3,3,2))

#extract second column from each U matrix
u12 = u1[:,1]
u22 = u2[:,1]
u32 = u3[:,1]
u42 = u4[:,1]

#perform outer product
for i in range(len(u42)):              #length should be 2 since there's 2␣
 ↪values in the columns of U4
    for j in range(len(u32)):          #length should be 3 since there's 3␣
 ↪values in the columns of U3
        for k in range(len(u22)):      #length should be 3 since there's 3␣
 ↪values in the columns of U2
            for l in range(len(u12)): #length should be 3 since there's 3␣
 ↪values in the columns of U1

                prod2[l][k][j][i] = u12[l] * u22[k] * u32[j] * u42[i]

#multiply outer product by the second lambda value
prod2 = prod2 * lam[1]
prod2
```

[5]:

```
array([[[[-0.18541814,  0.07893521],
          [-0.21475609,  0.0914248 ],
          [ 0.50959368, -0.21694147]],

         [[ 0.26959486, -0.11477047],
          [ 0.31225174, -0.13293012],
          [-0.74094064,  0.31542925]],
```

```
 [[ 0.32152295, -0.13687701],
  [ 0.3723962 , -0.1585345 ],
  [-0.88365713,  0.37618574]]],


[[[-1.23910818,  0.52750644],
  [-1.43516713,  0.6109716 ],
  [ 3.40550115, -1.44977155]],

 [[ 1.80164245, -0.76698549],
  [ 2.08670887, -0.88834242],
  [-4.95154139,  2.10794344]],

 [[ 2.14866631, -0.91471861],
  [ 2.48864088, -1.05945075],
  [-5.90528391,  2.51396556]]],


[[[ 0.15272699, -0.06501811],
  [ 0.17689235, -0.07530566],
  [-0.41974701,  0.17869243]],

 [[-0.22206248,  0.09453524],
  [-0.25719851,  0.10949316],
  [ 0.61030509, -0.25981579]],

 [[-0.26483511,  0.11274417],
  [-0.30673887,  0.13058321],
  [ 0.72785918, -0.30986028]]]])
```

0.1.3 χ the full reconstruction

[6]:
```
X = prod1 + prod2
X.shape
```

[6]:
```
(3, 3, 3, 2)
```

[7]:
```
X
```

[7]:
```
array([[[[4.65282255, 4.34319107],
         [4.03305524, 3.8352969 ],
         [6.25937447, 4.85071389]],

        [[6.20661977, 5.11791531],
         [5.52475835, 4.46119061],
         [6.31463882, 6.53396982]],
```

```
       [[5.85418015, 4.73941323],
        [5.22988167, 4.12268304],
        [5.69137045, 6.17118473]]],


      [[[3.73956755, 4.91553682],
        [2.93594139, 4.46351353],
        [9.32217538, 3.76497785]],

       [[7.91099576, 4.6175845 ],
        [7.45051398, 3.8391275 ],
        [2.30883371, 8.50698371]],

       [[7.84191472, 4.10311109],
        [7.48711999, 3.34603364],
        [0.86059077, 8.47717049]]],


      [[[4.98758371, 4.19625523],
        [4.42173266, 3.66594789],
        [5.32601224, 5.24280336]],

       [[5.71080993, 5.32356115],
        [4.95166236, 4.70040065],
        [7.66094972, 5.9543754 ]],

       [[5.26395243, 4.98562382],
        [4.54734917, 4.40880637],
        [7.29828804, 5.48108555]]]])
```

0.1.4 Use tensorly and compare results of manual calculation from library implementation

[8]:
```
X_tl = tl.kruskal_to_tensor((lam, [u1,u2,u3,u4]))
X_tl
```

[8]:
```
array([[[[4.65282255, 4.34319107],
         [4.03305524, 3.8352969 ],
         [6.25937447, 4.85071389]],

        [[6.20661977, 5.11791531],
         [5.52475835, 4.46119061],
         [6.31463882, 6.53396982]],

        [[5.85418015, 4.73941323],
```

```
  [5.22988167, 4.12268304],
  [5.69137045, 6.17118473]]],


[[[3.73956755, 4.91553682],
  [2.93594139, 4.46351353],
  [9.32217538, 3.76497785]],

 [[7.91099576, 4.6175845 ],
  [7.45051398, 3.8391275 ],
  [2.30883371, 8.50698371]],

 [[7.84191472, 4.10311109],
  [7.48711999, 3.34603364],
  [0.86059077, 8.47717049]]],


[[[4.98758371, 4.19625523],
  [4.42173266, 3.66594789],
  [5.32601224, 5.24280336]],

 [[5.71080993, 5.32356115],
  [4.95166236, 4.70040065],
  [7.66094972, 5.9543754 ]],

 [[5.26395243, 4.98562382],
  [4.54734917, 4.40880637],
  [7.29828804, 5.48108555]]]])
```

Part 2. A Tucker decomposition of the same original tensor is:

$$G_{1,1}=\begin{pmatrix}38.946 & 0.8653\\ 0.9666 & -4.8832\end{pmatrix} \quad G_{2,1}=\begin{pmatrix}-0.4799 & -0.0792\\ -1.7302 & -4.3675\end{pmatrix}$$

$$G_{1,2}=\begin{pmatrix}0.7059 & -1.6496\\ 0.7553 & -1.1648\end{pmatrix} \quad G_{2,2}=\begin{pmatrix}5.7493 & -3.3204\\ -2.0019 & 7.6587\end{pmatrix}$$

$$U_1=\begin{pmatrix}0.5661 & -0.1945\\ 0.6005 & -0.5685\\ 0.5648 & 0.7994\end{pmatrix} \quad U_2=\begin{pmatrix}0.5031 & 0.8331\\ 0.6345 & -0.1755\\ 0.5867 & -0.5246\end{pmatrix} \quad U_3=\begin{pmatrix}0.5773 & -0.3364\\ 0.5013 & -0.5733\\ 0.6445 & 0.7471\end{pmatrix}$$

$$U_4=\begin{pmatrix}0.7524 & -0.6587\\ 0.6587 & 0.7524\end{pmatrix}$$

Compute the reconstruction of the Tucker decomposition.

[9]:
```
g11 = np.array([[38.946, 0.8653],
                [0.9666, -4.8832]])

g12 = np.array([[0.7059, -1.6496],
                [0.7553, -1.1648]])

g21 = np.array([[-0.4799, -0.0792],
                [-1.7302, -4.3675]])

g22 = np.array([[5.7493, -3.3204],
                [-2.0019, 7.6587]])

g = np.zeros((2,2,2,2))


g[:,:,0,0] = g11
g[:,:,1,0] = g12
g[:,:,0,1] = g21
g[:,:,1,1] = g22

g = tl.tensor(g, dtype=tl.float32)
```

[10]:
```
u1 = np.array([[0.5661, -0.1945],
               [0.6005, -0.5685],
               [0.5648, 0.7994]])

u2 = np.array([[0.5031, 0.8331],
```

```
               [0.6345, -0.1755],
               [0.5867, -0.5246]])

u3 = np.array([[0.5773, -0.3364],
               [0.5013, -0.5733],
               [0.6445, 0.7471]])

u4 = np.array([[0.7524, -0.6587],
               [0.6587, 0.7524]])
```

0.1.5 Manual Calculation

[11]:
```
prod = []
for p in range(2):
    for q in range(2):
        for r in range(2):
            for s in range(2):

                prod.append(np.outer(np.outer(np.outer((g[p][q][r][s] * u1[:
 ↪,p]), u2[:,q]), u3[:,r]) , u4[:,s]))
```

[12]:
```
result = np.zeros(prod[0].shape)

for i in prod:
    result += i

result = result.reshape((3,3,3,2))
result
```

[12]:
```
array([[[[ 4.93169791,  5.29221616],
         [ 4.1998829 ,  4.88708718],
         [ 5.83543511,  4.74410601]],

        [[ 6.52414397,  4.33674962],
         [ 6.14284537,  3.09887053],
         [ 5.37677372,  7.50444614]],

        [[ 6.1225886 ,  3.3192871 ],
         [ 5.93006553,  1.95986379],
         [ 4.38583683,  7.388613  ]]],


       [[[ 4.68851114,  7.26166459],
         [ 3.57800094,  6.98710315],
         [ 7.20373333,  5.38633794]],
```

```
  [[ 6.91614031,  4.26511325],
   [ 6.74001646,  2.84167726],
   [ 4.7891852 ,  8.20297477]],

  [[ 6.69150931,  2.49794405],
   [ 6.89045511,  0.863456  ],
   [ 3.15897946,  8.00158859]]],


 [[[ 6.40945058,  0.76002819],
   [ 6.5960878 , -0.0698415 ],
   [ 3.04148456,  3.76233945]],

  [[ 6.52137912,  5.24614314],
   [ 5.51461142,  4.31375332],
   [ 7.87237581,  6.82201766]],

  [[ 5.56851484,  6.11785216],
   [ 4.27055427,  5.28945591],
   [ 8.47204259,  6.92181322]]]])
```

0.1.6 Tensorly Calculation

[13]:
```
result_tl = tl.tucker_to_tensor((g, [u1,u2,u3,u4]))
```

[14]:
```
result_tl
```

[14]:
```
array([[[[ 4.93169791,  5.29221616],
         [ 4.1998829 ,  4.88708718],
         [ 5.83543511,  4.74410601]],

        [[ 6.52414397,  4.33674962],
         [ 6.14284537,  3.09887053],
         [ 5.37677372,  7.50444614]],

        [[ 6.1225886 ,  3.3192871 ],
         [ 5.93006553,  1.95986379],
         [ 4.38583683,  7.388613  ]]],


       [[[ 4.68851114,  7.26166459],
         [ 3.57800094,  6.98710315],
         [ 7.20373333,  5.38633794]],

        [[ 6.91614031,  4.26511325],
         [ 6.74001646,  2.84167726],
```

```
 [ 4.7891852 ,  8.20297477]],

[[ 6.69150931,  2.49794405],
 [ 6.89045511,  0.863456  ],
 [ 3.15897946,  8.00158859]]],


[[[ 6.40945058,  0.76002819],
  [ 6.5960878 , -0.0698415 ],
  [ 3.04148456,  3.76233945]],

 [[ 6.52137912,  5.24614314],
  [ 5.51461142,  4.31375332],
  [ 7.87237581,  6.82201766]],

 [[ 5.56851484,  6.11785216],
  [ 4.27055427,  5.28945591],
  [ 8.47204259,  6.92181322]]]])
```

Part 3. The actual original tensor was:

$$X_{1,1}=\begin{pmatrix}4&0&9\\7&9&9\\4&8&5\end{pmatrix} X_{2,1}=\begin{pmatrix}7&8&2\\1&5&8\\7&9&2\end{pmatrix} X_{3,1}=\begin{pmatrix}7&9&4\\10&1&2\\1&5&8\end{pmatrix} X_{1,2}=\begin{pmatrix}6&5&1\\3&3&5\\1&8&7\end{pmatrix} X_{2,2}=\begin{pmatrix}8&2&3\\4&3&3\\2&4&6\end{pmatrix}$$

$$X_{3,2}=\begin{pmatrix}6&6&8\\5&9&8\\3&9&5\end{pmatrix}$$

Calculate the MSE for both the CP and Tucker decompositions. Briefly discuss (2-3 sentences should be sufficient) the difference, especially regarding the relative reduction of features for each method.

[15]:
```
x11 = np.array([[4,0,9],
       [7,9,9],
       [4,8,5]])

x21 = np.array([[7,8,2],
       [1,5,8],
       [7,9,2]])

x31 = np.array([[7,9,4],
       [10,1,2],
       [1,5,8]])

x12 = np.array([[6,5,1],
       [3,3,5],
       [1,8,7]])
```

```
x22 = np.array([[8,2,3],
       [4,3,3],
       [2,4,6]])

x32 = np.array([[6,6,8],
       [5,9,8],
       [3,9,5]])

x = np.zeros((3,3,3,2))


x[:,:,0,0] = x11
x[:,:,1,0] = x21
x[:,:,2,0] = x31
x[:,:,0,1] = x12
x[:,:,1,1] = x22
x[:,:,2,1] = x32
```

0.1.7 CP MSE

[16]:
```
((x - X_tl)**2).mean()
```

[16]: 5.033737515123824

0.2 Tucker MSE

[17]:
```
((x - result_tl)**2).mean()
```

[17]: 4.927798012475143

1 Result

The original Tensor had 54 parameters. After CP Decomposition, the tensor was reduced to 26 parameters from the orignal 54. From these parameters, we achieved a reconstruction MSE of ~5. Conversely after Tucker decomposition, we were left with 38 parameters, a reduction of 16 parameters. The reconstruction MSE was ~4.9. With ~0.1 differene in MSE, CP would be a better choice due to the fewer amount of parameters

[ ]:
```

```