UAS Artificial Neural Network

Name: Mohamad Dhiyaa Abhirama

NIM: 2001575630

Answers:

2. Dimension Reduction

In [1]:
```
import sys
import os
import pandas as pd
from sklearn.decomposition import PCA
import numpy.linalg as l
import numpy as np
from matplotlib import pyplot as plt
```

In [9]:
```
#Load dataset
def  load_dataset():
    df = pd.read_csv("D:\\Binus\\ANN\\UAS\\clustering.csv")
    features = df[[" sepal_length","sepal_width","petal_length","petal_width" ]]
    return features
```

In [3]: #apply PCA

```
def apply_pca(dataset):
    pca = PCA(n_components=2)#reducing the dimension to 2
    result = pca.fit_transform(dataset)
    return result
```

In [10]:

```
dataset = load_dataset()
print(dataset)
```

Out [10]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	7.0	3.2	4.7	1.4
11	6.4	3.2	4.5	1.5
12	6.9	3.1	4.9	1.5
13	5.5	2.3	4.0	1.3
14	6.5	2.8	4.6	1.5
15	5.7	2.8	4.5	1.3
16	6.3	3.3	4.7	1.6
17	4.9	2.4	3.3	1.0
18	6.6	2.9	4.6	1.3
19	5.2	2.7	3.9	1.4
20	6.3	3.3	6.0	2.5
21	5.8	2.7	5.1	1.9
22	7.1	3.0	5.9	2.1
23	6.3	2.9	5.6	1.8
24	6.5	3.0	5.8	2.2
25	7.6	3.0	6.6	2.1
26	4.9	2.5	4.5	1.7
27	7.3	2.9	6.3	1.8
28	6.7	2.5	5.8	1.8
29	7.2	3.6	6.1	2.5

In [11]: #compute sample covariance matrix

```
C = np.cov(dataset)

evals, evects = l.eig(C)

pca_dataset = apply_pca(dataset)

print(pca_dataset.shape) #see the shape after applying pca

print("after dimension reduction into two: ",pca_dataset)
```

Out [11]: (30, 2)

```
after dimension reduction into two:  [[-2.75962938 -0.48854573]
 [-2.81406061 -0.05403161]
 [-2.98791375 -0.03793733]
 [-2.85265071  0.14279299]
 [-2.80408874 -0.46850044]
 [-2.33668102 -0.830441  ]
 [-2.91773822 -0.02990435]
 [-2.70930076 -0.3318117 ]
 [-3.00679671  0.38427215]
 [-2.76868024 -0.10671916]
 [ 1.22463279 -0.73501648]
 [ 0.85219575 -0.30432241]
 [ 1.39452308 -0.53394854]
 [ 0.04681702  0.72966042]
 [ 0.99636989 -0.13133274]
 [ 0.52592032  0.41512778]
 [ 1.01223248 -0.21594691]
 [-0.89372792  0.9026763 ]
 [ 0.96172115 -0.30474571]
 [-0.14179501  0.72979138]
 [ 2.42802523  0.27904353]
 [ 1.28774975  0.66108599]
 [ 2.53562275 -0.26839665]
 [ 1.86560388  0.26607987]
 [ 2.24812013  0.18629372]
 [ 3.32782654 -0.48238012]
 [ 0.36294911  1.27664898]
 [ 2.85547056 -0.32986113]
 [ 2.21331016  0.23389132]
 [ 2.85397249 -0.5535224 ]]
```

In [12]:
```
#compute eigenvalues/eigenvectors using eig

print("eigen value: ",evals)

print("eigen vector: ",evects)
```

Out [12]:
```
eigen value:  [ 1.13014122e+02+0.00000000e+00j  1.93302665e+01+0.0
0000000e+00j
  1.60611586e-01+0.00000000e+00j  3.93510642e-15+0.00000000e+00j
 -2.97982398e-15+4.37270811e-16j -2.97982398e-15-4.37270811e-16j
  2.89410914e-15+0.00000000e+00j  2.18405435e-15+6.10515672e-16j
  2.18405435e-15-6.10515672e-16j  2.35090887e-15+0.00000000e+00j
 -2.17029456e-15+0.00000000e+00j -1.49615078e-15+8.69355924e-16j
 -1.49615078e-15-8.69355924e-16j -1.85828353e-15+0.00000000e+00j
 -1.45877052e-15+0.00000000e+00j -1.38437333e-15+0.00000000e+00j
  1.80194593e-15+0.00000000e+00j  1.74716500e-15+0.00000000e+00j
 -8.26953155e-16+2.40724304e-16j -8.26953155e-16-2.40724304e-16j
  1.29729687e-15+0.00000000e+00j -3.65715503e-16+2.19954936e-16j
 -3.65715503e-16-2.19954936e-16j  3.92699336e-16+3.41916494e-16j
  3.92699336e-16-3.41916494e-16j -1.83783868e-16+0.00000000e+00j
  1.35553452e-16+0.00000000e+00j  8.19416951e-16+0.00000000e+00j
  6.84548424e-16+0.00000000e+00j  4.24174448e-16+0.00000000e+00j]
eigen vector:  [[-1.67087242e-01+0.j         -2.87236670e-01+0.j
   9.13357598e-03+0.j          2.29796996e-01+0.j
   2.15801585e-02+0.15688103j  2.15801585e-02-0.15688103j
   1.17349702e-01+0.j          3.06313151e-02+0.05377136j
   3.06313151e-02-0.05377136j -1.09620996e-01+0.j
   1.23003742e-01+0.j         -5.67801800e-02+0.24057905j
  -5.67801800e-02-0.24057905j -1.25068125e-01+0.j
  -1.07647905e-01+0.j          6.73586634e-02+0.j
  -9.68495367e-02+0.j          9.99572403e-02+0.j
   7.22652028e-02+0.0244175j   7.22652028e-02-0.0244175j
   3.79909496e-02+0.j          2.52157517e-03-0.0404411j
   2.52157517e-03+0.0404411j   1.18274091e-01+0.07219929j
   1.18274091e-01-0.07219929j -1.06452858e-01+0.j
  -1.19374503e-02+0.j         -1.00023050e-01+0.j
  -1.48853060e-01+0.j         -1.37775657e-01+0.j        ]
 [-1.63829382e-01+0.j         -2.39116930e-01+0.j
   2.57085518e-01+0.j         -1.51120544e-01+0.j
  -1.49115775e-01+0.05141266j -1.49115775e-01-0.05141266j
   1.48850073e-02+0.j         -1.30807656e-02-0.06965809j
  -1.30807656e-02+0.06965809j -1.52528110e-02+0.j
   8.75036114e-02+0.j          7.36524694e-02+0.12164039j
   7.36524694e-02-0.12164039j -4.76737415e-02+0.j
   1.44167706e-02+0.j         -2.68427939e-02+0.j
  -1.03550984e-02+0.j          7.65400480e-03+0.j
   8.01624409e-02+0.01674075j  8.01624409e-02-0.01674075j
  -8.51081235e-03+0.j          6.73094232e-02-0.02099369j
   6.73094232e-02+0.02099369j  5.22160418e-02+0.02812966j
   5.22160418e-02-0.02812966j -2.62456648e-02+0.j
  -3.03634146e-02+0.j         -3.33470580e-02+0.j
```

```
 -4.20250114e-02+0.j         -3.15818784e-02+0.j        ]
[-1.53700003e-01+0.j         -2.61318334e-01+0.j
  3.33059243e-02+0.j          2.58276369e-01+0.j
 -3.17152741e-02-0.1782797j  -3.17152741e-02+0.1782797j
 -5.84441825e-03+0.j          2.96315358e-01+0.2545644j
  2.96315358e-01-0.2545644j  -1.89735532e-01+0.j
 -6.19423458e-02+0.j          2.15935619e-01-0.12146123j
  2.15935619e-01+0.12146123j  1.65659273e-01+0.j
  2.33802199e-01+0.j         -1.28870802e-01+0.j
  1.53524873e-01+0.j         -1.74355199e-01+0.j
 -3.66518314e-02+0.0260697j  -3.66518314e-02-0.0260697j
  3.23609680e-02+0.j          8.34616424e-02-0.05378735j
  8.34616424e-02+0.05378735j  4.59412608e-02-0.14275326j
  4.59412608e-02+0.14275326j  4.53269005e-02+0.j
  4.10741472e-02+0.j          1.08382742e-01+0.j
  6.07056237e-02+0.j         -7.58757350e-03+0.j        ]
[-1.52254596e-01+0.j         -2.31447115e-01+0.j
 -9.18468117e-02+0.j         -2.50682305e-01+0.j
  2.51192157e-02+0.14127926j  2.51192157e-02-0.14127926j
 -2.07339024e-01+0.j         -1.14915146e-01+0.03365647j
 -1.14915146e-01-0.03365647j  1.85149179e-01+0.j
  2.79612788e-02+0.j         -2.06678502e-01-0.04085895j
 -2.06678502e-01+0.04085895j -4.74564253e-02+0.j
 -1.67714711e-01+0.j          3.74273948e-01+0.j
  1.28122424e-01+0.j         -1.32599294e-01+0.j
 -4.12644574e-01+0.j         -4.12644574e-01-0.j
 -3.34746604e-02+0.j         -1.79029676e-01+0.0778582j
 -1.79029676e-01-0.0778582j  -2.21105510e-01-0.04076505j
 -2.21105510e-01+0.04076505j -1.46120243e-01+0.j
 -8.14799624e-02+0.j          2.03634741e-01+0.j
  3.55143802e-01+0.j          1.23205913e-01+0.j        ]
[-1.62396486e-01+0.j         -2.93622496e-01+0.j
 -1.36185370e-01+0.j          1.62687737e-02+0.j
 -3.17767950e-02-0.0615564j  -3.17767950e-02+0.0615564j
  1.14856107e-01+0.j         -1.87859659e-01-0.06941513j
 -1.87859659e-01+0.06941513j  9.98759889e-02+0.j
 -1.92278432e-02+0.j          1.14586811e-01-0.0315469j
  1.14586811e-01+0.0315469j  -1.18664107e-02+0.j
  6.04743743e-02+0.j         -1.62250186e-01+0.j
  1.34754105e-01+0.j         -1.17486189e-01+0.j
  6.99207866e-02-0.01328033j  6.99207866e-02+0.01328033j
 -9.13521894e-02+0.j         -2.04810321e-01+0.02936108j
 -2.04810321e-01-0.02936108j -5.74990958e-02-0.07300171j
 -5.74990958e-02+0.07300171j  3.47144425e-02+0.j
  1.01300885e-01+0.j          2.01427914e-01+0.j
 -3.27339997e-02+0.j          2.31313628e-01+0.j        ]
[-1.69901443e-01+0.j         -2.97517403e-01+0.j
 -1.43229992e-01+0.j          7.82535693e-02+0.j
 -1.07697175e-02+0.09430114j -1.07697175e-02-0.09430114j
 -7.35646917e-02+0.j          1.39716569e-01-0.23976266j
  1.39716569e-01+0.23976266j -1.12868959e-01+0.j
 -1.36280104e-01+0.j         -3.33203852e-01-0.03571267j
 -3.33203852e-01+0.03571267j  7.86809824e-02+0.j
 -1.42242731e-01+0.j          2.58765530e-01+0.j
```

```
 -5.84434026e-02+0.j          5.43903305e-02+0.j
 -2.67155800e-01-0.05697555j -2.67155800e-01+0.05697555j
 -8.67679203e-02+0.j         -2.71474644e-01+0.12065602j
 -2.71474644e-01-0.12065602j  1.03104987e-02-0.16157518j
  1.03104987e-02+0.16157518j  3.49112623e-01+0.j
  4.23081091e-01+0.j          1.16179106e-01+0.j
 -8.16503545e-02+0.j          4.23229939e-02+0.j        ]
[-1.45193298e-01+0.j         -2.65391215e-01+0.j
 -1.80390564e-01+0.j         -9.66164655e-02+0.j
  5.65795450e-03-0.02891483j  5.65795450e-03+0.02891483j
 -6.97465372e-02+0.j         -2.65701905e-02+0.04261136j
 -2.65701905e-02-0.04261136j  1.02938329e-01+0.j
  2.73719386e-02+0.j          4.34647434e-02+0.01701781j
  4.34647434e-02-0.01701781j -6.47266870e-02+0.j
 -1.07569184e-01+0.j         -1.28009506e-01+0.j
  3.67839103e-02+0.j         -4.54707280e-02+0.j
  2.10740931e-01+0.01357623j  2.10740931e-01-0.01357623j
 -9.64096416e-02+0.j         -5.88504286e-02+0.08349658j
 -5.88504286e-02-0.08349658j  1.51061659e-02+0.09799042j
  1.51061659e-02-0.09799042j  4.64157498e-02+0.j
 -4.31245106e-03+0.j         -2.19569388e-01+0.j
 -1.63140238e-01+0.j          1.15163717e-01+0.j        ]
[-1.64893968e-01+0.j         -2.66795203e-01+0.j
 -4.91608604e-02+0.j         -1.05847817e-01+0.j
  5.50830094e-02-0.07473378j  5.50830094e-02+0.07473378j
 -8.26729810e-02+0.j         -2.48669036e-02+0.01799385j
 -2.48669036e-02-0.01799385j  1.00813658e-01+0.j
  6.75015209e-02+0.j          3.17108649e-02-0.11595225j
  3.17108649e-02+0.11595225j -1.16565146e-01+0.j
 -2.52758107e-02+0.j          1.58240744e-02+0.j
 -1.55860938e-01+0.j          1.95567752e-01+0.j
  2.08726023e-01+0.01033934j  2.08726023e-01-0.01033934j
  3.66878319e-01+0.j          2.40826152e-02-0.07343969j
  2.40826152e-02+0.07343969j  2.17489952e-01-0.08584033j
  2.17489952e-01+0.08584033j  2.27895187e-02+0.j
  1.42757361e-01+0.j          1.41762634e-01+0.j
 -1.56010203e-01+0.j         -1.45545987e-01+0.j        ]
[-1.45624446e-01+0.j         -2.18853437e-01+0.j
 -7.86078522e-03+0.j          8.11698160e-02+0.j
 -2.33249302e-01-0.07374441j -2.33249302e-01+0.07374441j
 -1.03165688e-01+0.j          5.88585656e-02+0.08839512j
  5.88585656e-02-0.08839512j -5.91403042e-02+0.j
  2.64961277e-02+0.j         -1.02638498e-01+0.14985779j
 -1.02638498e-01-0.14985779j  4.56348841e-02+0.j
  1.49941025e-02+0.j         -5.32306481e-03+0.j
 -1.56414351e-01+0.j          1.60475272e-01+0.j
  9.72361091e-03+0.05391716j  9.72361091e-03-0.05391716j
  1.81858066e-01+0.j          2.08513119e-01-0.0214974j
  2.08513119e-01+0.0214974j  -1.36721033e-01+0.23158275j
 -1.36721033e-01-0.23158275j  8.65393937e-03+0.j
 -4.37748441e-01+0.j          6.02081157e-02+0.j
  1.23263264e-01+0.j          1.34937151e-01+0.j        ]
[-1.67391450e-01+0.j         -2.39967910e-01+0.j
  3.78636498e-02+0.j         -1.84464475e-01+0.j
```

```
  2.00253185e-01-0.0038683j   2.00253185e-01+0.0038683j
  1.24815523e-01+0.j         -1.59857984e-01-0.07417471j
 -1.59857984e-01+0.07417471j  1.27090339e-02+0.j
 -3.92135244e-02+0.j          4.49028180e-02-0.23380876j
  4.49028180e-02+0.23380876j -3.59741690e-02+0.j
  3.12951302e-02+0.j         -1.46132365e-01+0.j
  2.49905296e-01+0.j         -2.78440260e-01+0.j
  6.23604012e-02-0.02890605j  6.23604012e-02+0.02890605j
 -3.95769083e-01+0.j          2.36906575e-01-0.11051424j
  2.36906575e-01+0.11051424j -3.63609486e-02+0.05677025j
 -3.63609486e-02-0.05677025j -1.29470952e-01+0.j
 -1.62030413e-01+0.j         -1.27935868e-01+0.j
  1.68291966e-01+0.j          7.65414346e-02+0.j        ]
[-2.22626202e-01+0.j          2.50790682e-02+0.j
  2.37873146e-01+0.j         -5.31656331e-02+0.j
  2.49969098e-01+0.15211685j  2.49969098e-01-0.15211685j
  1.88018786e-01+0.j          5.13796548e-02-0.0286856j
  5.13796548e-02+0.0286856j  -1.38214875e-01+0.j
 -5.93820591e-01+0.j          2.38706658e-01-0.22383002j
  2.38706658e-01+0.22383002j  6.24626735e-01+0.j
  4.22498939e-01+0.j         -2.21295000e-01+0.j
 -1.25822649e-01+0.j          1.18816611e-01+0.j
  2.06384516e-01+0.06865374j  2.06384516e-01-0.06865374j
 -8.44275667e-02+0.j          2.39663485e-01-0.16795524j
  2.39663485e-01+0.16795524j  3.85511927e-02+0.05481926j
  3.85511927e-02-0.05481926j -2.42254204e-01+0.j
 -1.18879163e-01+0.j         -1.00350538e-01+0.j
 -1.51043104e-01+0.j         -1.31488186e-02+0.j        ]
[-1.94545768e-01+0.j          2.16791930e-02+0.j
  3.67466596e-02+0.j         -1.00846279e-01+0.j
 -4.66405481e-02-0.0489207j  -4.66405481e-02+0.0489207j
 -1.92502956e-01+0.j          5.60209755e-03-0.00987854j
  5.60209755e-03+0.00987854j -1.40870718e-02+0.j
 -2.21569533e-02+0.j          9.78647364e-03+0.06703958j
  9.78647364e-03-0.06703958j -2.83248030e-02+0.j
  1.13842343e-01+0.j         -1.30363767e-01+0.j
 -1.93315666e-01+0.j          1.81754082e-01+0.j
 -1.69292583e-01+0.01041237j -1.69292583e-01-0.01041237j
  4.00292964e-01+0.j         -9.61343487e-02-0.10712716j
 -9.61343487e-02+0.10712716j -1.34104603e-01+0.00956734j
 -1.34104603e-01-0.00956734j -3.03620851e-01+0.j
  1.15185615e-02+0.j         -1.74632244e-01+0.j
  1.64073065e-01+0.j         -2.81185592e-01+0.j        ]
[-2.17491787e-01+0.j          5.65322497e-02+0.j
  1.88142573e-01+0.j         -4.88111007e-03+0.j
  6.27098491e-02-0.10528118j  6.27098491e-02+0.10528118j
  1.36098412e-01+0.j          2.64053313e-03-0.06215431j
  2.64053313e-03+0.06215431j  4.72311291e-03+0.j
  3.30275116e-01+0.j          4.45838758e-02+0.17415646j
  4.45838758e-02-0.17415646j -2.62972801e-01+0.j
 -8.68879137e-02+0.j          4.31724813e-02+0.j
 -3.62486993e-02+0.j          5.86057395e-02+0.j
 -1.74141169e-01-0.0031666j  -1.74141169e-01+0.0031666j
 -3.02823294e-01+0.j          4.60853773e-02+0.05082785j
```

```
  4.60853773e-02-0.05082785j  3.36450930e-02+0.11203886j
  3.36450930e-02-0.11203886j -1.12704663e-01+0.j
 -3.13447776e-02+0.j          1.81769658e-01+0.j
 -9.66797321e-02+0.j         -3.13920533e-02+0.j        ]
[-1.71714179e-01+0.j          7.04719395e-02+0.j
  2.97268603e-01+0.j          1.22013127e-01+0.j
  1.26614673e-01+0.10730107j  1.26614673e-01-0.10730107j
 -5.40261414e-02+0.j         -6.53044733e-02-0.0028176j
 -6.53044733e-02+0.0028176j   2.74958008e-01+0.j
 -1.92616347e-01+0.j          7.15795648e-02-0.01019721j
  7.15795648e-02+0.01019721j  2.11943279e-01+0.j
  2.66486229e-01+0.j         -3.29462871e-01+0.j
  1.83625447e-01+0.j         -1.45855206e-01+0.j
  1.88347365e-01-0.02529883j  1.88347365e-01+0.02529883j
  2.31936680e-01+0.j         -1.98403137e-02+0.19726638j
 -1.98403137e-02-0.19726638j -8.25146724e-02+0.0302495j
 -8.25146724e-02-0.0302495j   2.75104679e-01+0.j
 -2.53978674e-02+0.j          6.54772495e-02+0.j
  1.19963577e-01+0.j          2.83588792e-01+0.j        ]
[-2.02608814e-01+0.j          6.35910819e-02+0.j
  3.46031521e-01+0.j          5.75175489e-02+0.j
 -8.56703462e-02-0.0155744j  -8.56703462e-02+0.0155744j
  8.28375506e-02+0.j         -2.18985501e-01+0.12285299j
 -2.18985501e-01-0.12285299j  2.74221547e-01+0.j
  9.74498768e-02+0.j         -6.27108603e-02+0.08102914j
 -6.27108603e-02-0.08102914j -1.62834271e-01+0.j
 -1.59309081e-01+0.j          2.08885077e-01+0.j
  1.03363003e-01+0.j         -1.26169385e-01+0.j
 -5.70813099e-02+0.03540525j -5.70813099e-02-0.03540525j
  5.69107032e-02+0.j          2.96778312e-02+0.15314606j
  2.96778312e-02-0.15314606j -2.64008570e-02-0.08935011j
 -2.64008570e-02+0.08935011j  3.62436320e-01+0.j
  6.41842688e-02+0.j         -6.59160652e-02+0.j
 -8.23048997e-02+0.j          6.26951396e-02+0.j        ]
[-1.78711374e-01+0.j          6.95009598e-02+0.j
 -2.84974838e-01+0.j          6.18588183e-02+0.j
 -1.02621769e-01+0.0228321j  -1.02621769e-01-0.0228321j
  4.36847595e-01+0.j          3.44092904e-02+0.16004522j
  3.44092904e-02-0.16004522j -7.54022329e-02+0.j
 -6.02185810e-02+0.j         -5.64731593e-02-0.12174081j
 -5.64731593e-02+0.12174081j  5.00980481e-02+0.j
 -1.05927113e-01+0.j          2.68401182e-02+0.j
 -2.88656409e-02+0.j          3.38518225e-02+0.j
 -1.25301271e-01-0.11548607j -1.25301271e-01+0.11548607j
  1.95483328e-01+0.j          3.68815475e-02+0.12197426j
  3.68815475e-02-0.12197426j  1.04139389e-01-0.13221987j
  1.04139389e-01+0.13221987j  3.57272468e-01+0.j
  2.70531463e-01+0.j          1.58845534e-01+0.j
 -4.94961672e-02+0.j          9.92681106e-02+0.j        ]
[-1.87661738e-01+0.j          3.57348339e-02+0.j
 -1.66866723e-01+0.j          2.59683928e-01+0.j
 -2.24125998e-01-0.16552646j -2.24125998e-01+0.16552646j
  2.85250341e-01+0.j         -4.95762958e-02-0.00413557j
 -4.95762958e-02+0.00413557j  2.48135466e-02+0.j
```

```
  2.33771583e-01+0.j         -4.40922596e-02+0.28821408j
 -4.40922596e-02-0.28821408j -1.03487805e-01+0.j
 -6.02448246e-03+0.j          2.18829243e-01+0.j
 -1.11663180e-01+0.j          1.23441157e-01+0.j
 -2.47347739e-02+0.09175666j -2.47347739e-02-0.09175666j
 -1.41260761e-01+0.j          4.93475416e-02-0.03401512j
  4.93475416e-02+0.03401512j  4.28765876e-03+0.08387256j
  4.28765876e-03-0.08387256j -2.21006905e-01+0.j
 -1.52603901e-01+0.j         -4.56411297e-01+0.j
 -9.31892329e-02+0.j         -1.25593610e-01+0.j        ]
[-1.53775635e-01+0.j          4.89668302e-03+0.j
  5.18035571e-02+0.j          1.76391395e-01+0.j
  2.35347007e-03-0.03509409j  2.35347007e-03+0.03509409j
  4.40119174e-02+0.j          1.06488940e-02-0.07588941j
  1.06488940e-02+0.07588941j -2.85801272e-02+0.j
 -4.14093524e-02+0.j          1.68387486e-01-0.10055751j
  1.68387486e-01+0.10055751j  1.25972011e-01+0.j
  2.00329766e-01+0.j         -3.31864465e-01+0.j
 -2.55229138e-01+0.j          2.75606065e-01+0.j
 -6.06839096e-02-0.07115116j -6.06839096e-02+0.07115116j
  2.33041625e-01+0.j          1.04816096e-01-0.04683537j
  1.04816096e-01+0.04683537j  4.28304946e-01+0.j
  4.28304946e-01-0.j         -6.90192895e-03+0.j
  9.97322816e-02+0.j         -2.97041965e-01+0.j
 -3.67592787e-01+0.j         -6.29274856e-01+0.j        ]
[-2.12927971e-01+0.j          4.94154420e-02+0.j
  1.86623332e-01+0.j         -1.31472059e-01+0.j
  6.18568746e-02+0.00384918j  6.18568746e-02-0.00384918j
 -3.56437655e-01+0.j          1.45447465e-01+0.05284623j
  1.45447465e-01-0.05284623j -1.22582686e-01+0.j
  5.23048919e-02+0.j         -2.03649298e-02-0.0071989j
 -2.03649298e-02+0.0071989j  -6.17100571e-02+0.j
 -6.08659000e-03+0.j         -2.12644088e-02+0.j
 -1.42840744e-01+0.j          1.14137805e-01+0.j
  8.02794313e-03-0.13896015j  8.02794313e-03+0.13896015j
 -2.25137215e-01+0.j         -7.71605880e-02+0.00539818j
 -7.71605880e-02-0.00539818j -2.56503753e-01-0.02916584j
 -2.56503753e-01+0.02916584j  4.57546107e-03+0.j
 -8.85107749e-02+0.j         -1.79878257e-01+0.j
  2.31224756e-01+0.j         -1.08132484e-01+0.j        ]
[-1.52330228e-01+0.j          3.47679020e-02+0.j
 -7.33491789e-02+0.j          7.82540921e-02+0.j
  1.46273015e-01-0.00310544j  1.46273015e-01+0.00310544j
  1.61041835e-01+0.j          1.13363421e-01-0.0348232j
  1.13363421e-01+0.0348232j  -1.79141308e-01+0.j
  4.52096504e-03+0.j          1.47541970e-01+0.09113527j
  1.47541970e-01-0.09113527j  2.71206314e-02+0.j
  5.40362358e-02+0.j          7.86497116e-02+0.j
  6.06541548e-02+0.j         -8.28738509e-02+0.j
  2.32954581e-01-0.02246876j  2.32954581e-01+0.02246876j
  7.70233473e-02+0.j          6.66462327e-02+0.04511171j
  6.66462327e-02-0.04511171j -3.57037030e-01+0.15311261j
 -3.57037030e-01-0.15311261j -2.71613750e-01+0.j
 -3.54870203e-01+0.j         -7.59714450e-02+0.j
```

```
  4.20538673e-01+0.j         -2.35265283e-01+0.j        ]
 [-1.64182936e-01+0.j          1.72559275e-01+0.j
 -3.57225152e-01+0.j         -1.10074049e-01+0.j
  1.63791584e-01+0.03249466j  1.63791584e-01-0.03249466j
  2.82375164e-02+0.j         -6.96407082e-02-0.07435037j
 -6.96407082e-02+0.07435037j -1.39879530e-02+0.j
 -9.36364286e-03+0.j          5.56081527e-02+0.0513749j
  5.56081527e-02-0.0513749j   8.63741879e-02+0.j
  2.51447496e-01+0.j         -1.50046593e-01+0.j
 -7.11195585e-02+0.j          4.74481755e-02+0.j
 -8.06004068e-02-0.14238878j -8.06004068e-02+0.14238878j
 -9.33333527e-03+0.j         -2.30015757e-01+0.03699644j
 -2.30015757e-01-0.03699644j  9.48272683e-02-0.06540766j
  9.48272683e-02+0.06540766j  4.82090477e-02+0.j
  1.78809222e-01+0.j          2.61653826e-01+0.j
 -1.90222319e-01+0.j          2.26351244e-01+0.j        ]
 [-1.65755283e-01+0.j          1.41957074e-01+0.j
 -8.82727109e-02+0.j         -1.61115683e-01+0.j
 -2.13524171e-01-0.06865634j -2.13524171e-01+0.06865634j
 -1.55003380e-01+0.j         -2.96452744e-01+0.0626773j
 -2.96452744e-01-0.0626773j   4.25433426e-01+0.j
  2.86663025e-01+0.j         -1.10425718e-01-0.03781668j
 -1.10425718e-01+0.03781668j -1.71394777e-01+0.j
 -1.88260091e-02+0.j          1.12400089e-01+0.j
  3.73278695e-01+0.j         -3.50559847e-01+0.j
 -3.84555176e-02+0.08046352j -3.84555176e-02-0.08046352j
 -2.62982569e-02+0.j         -3.60105446e-02-0.17193174j
 -3.60105446e-02+0.17193174j -4.15789571e-02+0.06822786j
 -4.15789571e-02-0.06822786j -1.05987951e-01+0.j
 -9.75146054e-02+0.j         -5.43657654e-02+0.j
  1.34551914e-02+0.j         -3.05372376e-02+0.j        ]
 [-2.11343113e-01+0.j          1.64394428e-01+0.j
  1.85789043e-01+0.j          2.76353967e-02+0.j
  2.34166284e-02-0.07323818j  2.34166284e-02+0.07323818j
 -1.34024330e-01+0.j         -8.50021938e-02-0.20297299j
 -8.50021938e-02+0.20297299j  8.51103684e-02+0.j
 -1.46113025e-01+0.j          7.56861231e-03-0.02766185j
  7.56861231e-03+0.02766185j  8.58734514e-02+0.j
 -1.39231022e-01+0.j          2.17225896e-01+0.j
 -1.53773268e-01+0.j          1.73933501e-01+0.j
 -9.39451704e-02-0.09655091j -9.39451704e-02+0.09655091j
  7.22950903e-02+0.j         -2.45934596e-01-0.06678193j
 -2.45934596e-01+0.06678193j  2.68159509e-01-0.01011444j
  2.68159509e-01+0.01011444j -2.90429435e-02+0.j
  3.08698766e-01+0.j          2.98753577e-01+0.j
 -2.22529104e-01+0.j          2.09721849e-01+0.j        ]
 [-1.90513755e-01+0.j          1.58561609e-01+0.j
 -3.09981475e-01+0.j          2.11251194e-01+0.j
 -1.47446452e-01+0.07071762j -1.47446452e-01-0.07071762j
 -3.37273049e-01+0.j          1.36432725e-01-0.02621037j
  1.36432725e-01+0.02621037j  8.19103184e-03+0.j
  5.32734738e-03+0.j          5.37190605e-02+0.01669236j
  5.37190605e-02-0.01669236j -1.32903870e-01+0.j
 -1.45563142e-01+0.j          1.96776361e-02+0.j
```

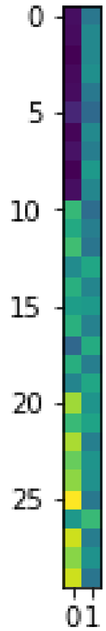
```
 -8.53552317e-02+0.j          8.77583890e-02+0.j
  1.42774888e-03-0.01203828j  1.42774888e-03+0.01203828j
 -6.12205402e-02+0.j          3.30318998e-01+0.j
  3.30318998e-01-0.j         -3.51681789e-02+0.0556517j
 -3.51681789e-02-0.0556517j  -9.33908806e-02+0.j
 -1.84608567e-01+0.j         -1.67599963e-01+0.j
 -7.80386659e-02+0.j         -4.72539815e-02+0.j        ]
[-1.84010546e-01+0.j          1.70424305e-01+0.j
 -8.21957437e-02+0.j          1.77126816e-01+0.j
  1.06110806e-01+0.06602072j  1.06110806e-01-0.06602072j
  2.78853768e-01+0.j          1.51713021e-01-0.13602936j
  1.51713021e-01+0.13602936j -2.91054330e-02+0.j
 -1.21334647e-02+0.j         -4.90667053e-02-0.09215638j
 -4.90667053e-02+0.09215638j -1.09964632e-01+0.j
 -1.95567883e-01+0.j          5.05296066e-02+0.j
 -6.41679631e-02+0.j          6.86570609e-02+0.j
  7.32073436e-02+0.02673761j  7.32073436e-02-0.02673761j
  1.47711646e-01+0.j         -7.29372725e-02-0.18487338j
 -7.29372725e-02+0.18487338j -1.50993363e-01-0.03847155j
 -1.50993363e-01+0.03847155j -2.07535705e-01+0.j
 -1.51650590e-01+0.j          3.12397445e-01+0.j
  2.95234052e-01+0.j          1.13518707e-01+0.j        ]
[-2.35657926e-01+0.j          2.18837971e-01+0.j
  5.96686523e-02+0.j         -1.93451726e-01+0.j
 -2.50186048e-01+0.0128301j  -2.50186048e-01-0.0128301j
  1.21078616e-01+0.j          1.14844528e-03+0.15833099j
  1.14844528e-03-0.15833099j -1.46492704e-01+0.j
  8.86734808e-03+0.j         -2.46439142e-01+0.07865728j
 -2.46439142e-01-0.07865728j -1.45611638e-01+0.j
 -1.68059351e-01+0.j          1.87097756e-01+0.j
  2.98439476e-01+0.j         -2.86581697e-01+0.j
 -2.98929783e-01-0.05355629j -2.98929783e-01+0.05355629j
 -1.94351448e-01+0.j         -2.73880010e-01+0.11920649j
 -2.73880010e-01-0.11920649j  7.89475183e-02-0.09148666j
  7.89475183e-02+0.09148666j  1.93012504e-01+0.j
  2.46253772e-01+0.j         -6.12732413e-02+0.j
 -1.56810201e-01+0.j         -9.18583228e-03+0.j        ]
[-1.36052174e-01+0.j          1.20428676e-01+0.j
 -2.99482303e-01+0.j          9.13850333e-02+0.j
  8.11455703e-02+0.03796446j  8.11455703e-02-0.03796446j
 -5.65508538e-02+0.j          8.07545288e-02+0.04145732j
  8.07545288e-02-0.04145732j -9.92325493e-02+0.j
 -2.81099063e-01+0.j          3.39271137e-01+0.j
  3.39271137e-01-0.j          3.06599062e-01+0.j
  4.34882842e-01+0.j         -2.85431227e-01+0.j
 -1.85104180e-01+0.j          1.88623592e-01+0.j
  3.47366812e-01+0.18940212j  3.47366812e-01-0.18940212j
  2.57431172e-02+0.j          2.56544175e-01+0.00507129j
  2.56544175e-01-0.00507129j  1.49035742e-01+0.19688992j
  1.49035742e-01-0.19688992j  2.07448726e-01+0.j
 -1.87757064e-01+0.j         -2.00776942e-01+0.j
 -6.78440819e-02+0.j         -5.62683145e-02+0.j        ]
[-2.33908311e-01+0.j          2.01440430e-01+0.j
 -9.42141576e-02+0.j         -4.00936340e-01+0.j
```

```
  1.31633559e-01+0.00670629j  1.31633559e-01-0.00670629j
 -2.32615934e-01+0.j         -2.93564670e-01+0.08813501j
 -2.93564670e-01-0.08813501j  3.66878927e-01+0.j
 -2.47820903e-01+0.j         -4.09110647e-02-0.03631382j
 -4.09110647e-02+0.03631382j  2.85887673e-01+0.j
  1.41575153e-01+0.j         -2.44149585e-01+0.j
  4.43025535e-01+0.j         -4.48038600e-01+0.j
  1.62130620e-01+0.0467609j   1.62130620e-01-0.0467609j
 -2.53004853e-01+0.j         -1.93067015e-04+0.01466503j
 -1.93067015e-04-0.01466503j -1.48790751e-01-0.10004876j
 -1.48790751e-01+0.10004876j -1.20628260e-01+0.j
 -3.97201941e-02+0.j          1.05993057e-01+0.j
  2.73194518e-01+0.j         -1.53899480e-01+0.j        ]
[-2.10772513e-01+0.j          2.02964417e-01+0.j
  1.37577712e-01+0.j          4.31206851e-01+0.j
 -3.52818198e-01-0.02563921j -3.52818198e-01+0.02563921j
  1.27571964e-01+0.j          4.11178098e-01+0.j
  4.11178098e-01-0.j         -5.36156941e-01+0.j
  2.74750423e-01+0.j         -1.40230546e-01-0.06368622j
 -1.40230546e-01+0.06368622j -2.41190190e-01+0.j
 -1.17604029e-01+0.j          3.84077302e-02+0.j
 -3.39542355e-01+0.j          3.07813424e-01+0.j
  1.31528436e-01+0.11570782j  1.31528436e-01-0.11570782j
  5.16425086e-02+0.j         -6.84557225e-04+0.00262691j
 -6.84557225e-04-0.00262691j  5.25125909e-02+0.02785113j
  5.25125909e-02-0.02785113j  2.63455631e-02+0.j
 -2.84304437e-02+0.j          8.62387170e-02+0.j
 -4.92467725e-02+0.j          2.34483211e-01+0.j        ]
[-1.96649918e-01+0.j          1.35076216e-01+0.j
 -3.95097923e-02+0.j         -2.50920631e-01+0.j
  4.22077405e-01+0.j          4.22077405e-01-0.j
 -1.89953702e-01+0.j         -3.54769758e-02-0.06579391j
 -3.54769758e-02+0.06579391j -5.70867043e-02+0.j
  2.34054264e-01+0.j         -1.52016517e-02-0.07229163j
 -1.52016517e-02+0.07229163j -1.75058480e-01+0.j
 -3.53144102e-01+0.j          2.41920641e-01+0.j
  2.47057606e-02+0.j         -2.42172173e-02+0.j
 -6.64496979e-02+0.0080651j  -6.64496979e-02-0.0080651j
  1.57996814e-01+0.j          1.01567137e-01+0.03453141j
  1.01567137e-01-0.03453141j  6.27381576e-03-0.08602537j
  6.27381576e-03+0.08602537j  1.98245148e-01+0.j
  1.03627803e-01+0.j         -4.54231942e-02+0.j
  4.29289640e-02+0.j          6.74601245e-02+0.j        ]]
```

In [13]: #showing the visualization after the dimension is reduced to 2

```
plt.imshow(pca_dataset)

plt.show()
```

Out [13]:



1. SOM (cluster)

In [1]:
```
from __future__ import division

import numpy as np
from matplotlib import pyplot as plt
from matplotlib import patches as patches
```

In [8]:
```
#set weight
w1 = np.array([np.ones(1)*1, np.ones(1)*1,np.ones(1)*-1])
w2 = np.array([np.ones(1)*2, np.ones(1)*1,np.ones(1)*1])
w3 = np.array([np.ones(1)*-1, np.ones(1)*2,np.ones(1)*-3])
w4 = np.array([np.ones(1)*1, np.ones(1)*2,np.ones(1)*3])
w5 = np.array([np.ones(1)*1, np.ones(1)*1,np.ones(1)*3])
w1 = np.hstack(w1)
w2 = np.hstack(w2)
w3 = np.hstack(w3)
w4 = np.hstack(w4)
w5 = np.hstack(w5)
weight = np.array((w1,w2,w3,w4,w5))
#weight = weight.transpose()
```

Out [8]:

```
[[ 1.  1. -1.]
 [ 2.  1.  1.]
 [-1.  2. -3.]
 [ 1.  2.  3.]
 [ 1.  1.  3.]]
```

In [22]:
```
# create a dataset with 2 clusters and 2 features
raw_data1=np.array([np.ones(1)*1,np.ones(1)*2,np.ones(1)*-1])
raw_data2=np.array([np.ones(1)*-1,np.ones(1)*3,np.ones(1)*-2])
raw_data=np.hstack((raw_data1, raw_data2))
raw_data=(raw_data + weight*0.2)
```

In [29]:
```
# create map dimension

network_dimensions = np.array([5, 5])

n_iterations = 1000
init_learning_rate = 0.5

normalise_data = True

# if True, assume all data on common scale
# if False, normalise to [0 1] range along each column
normalise_by_column = False

# establish variables based on data
m = raw_data.shape[0]
n = raw_data.shape[1]

# initial neighbourhood radius
init_radius = max(network_dimensions[0], network_dimensions[1]) / 2
# radius decay parameter
time_constant = n_iterations / np.log(init_radius)

data = raw_data
```

In [50]:
```
# check if data needs to be normalised
if normalise_data:
    if normalise_by_column:
        # normalise along each column
        col_maxes = raw_data.max(axis=0)
        data = raw_data / col_maxes[np.newaxis, :]
    else:
        # normalise entire dataset
        data = raw_data / data.max()

# setup random weights between 0 and 1
# weight matrix needs to be one m-dimensional vector for each neuron in the SOM
net = np.random.random((network_dimensions[0], network_dimensions[1], m))
```

In [50]: #find best matching unit or neuron winner, calculating Euclidean distance to find the nearest path to the x neuron.

```
def find_bmu(t, net, m):
    """
        Find the best matching unit for a given vector, t, in the SOM
        Returns: a (bmu, bmu_idx) tuple where bmu is the high-dimensional BMU
                 and bmu_idx is the index of this vector in the SOM
    """
    bmu_idx = np.array([0, 0])
    # set the initial minimum distance to a huge number
    min_dist = np.iinfo(np.int).max
    # calculate the high-dimensional distance between each neuron and the input
    for x in range(net.shape[0]):
        for y in range(net.shape[1]):
            w = net[x, y, :].reshape(m, 1)
            # don't bother with actual Euclidean distance, to avoid expensive
sqrt operation
            sq_dist = np.sum((w - t) ** 2)
            if sq_dist < min_dist:
                min_dist = sq_dist
                bmu_idx = np.array([x, y])
    # get vector corresponding to bmu_idx
    bmu = net[bmu_idx[0], bmu_idx[1], :].reshape(m, 1)
    # return the (bmu, bmu_idx) tuple
    return (bmu, bmu_idx)
```

In [89]: #decay radius

```
def decay_radius(initial_radius, i, time_constant):
    return initial_radius * np.exp(-i / time_constant)
```

In [93]: #decay learning rate

```
def decay_learning_rate(initial_learning_rate, i, n_iterations):
    return initial_learning_rate * np.exp(-i / n_iterations)
```

In [97]: #calculate for neighbour strength

```
def calculate_influence(distance, radius):
    return np.exp(-distance / (2* (radius**2)))


for i in range(n_iterations):
    #print('Iteration %d' % i)

    # select a training example at random
    t = data[:, np.random.randint(0, n)].reshape(np.array([m, 1]))

    # find its Best Matching Unit
    bmu, bmu_idx = find_bmu(t, net, m)

    #init
    r = 1
    l = 1

    # now we know the BMU, update its weight vector to move closer to input
    # and move its neighbours in 2-D space closer
    # by a factor proportional to their 2-D distance from the BMU
    for x in range(net.shape[0]):
        for y in range(net.shape[1]):
            w = net[x, y, :].reshape(m, 1)
            # get the 2-D distance (again, not the actual Euclidean distance)
            w_dist = np.sum((np.array([x, y]) - bmu_idx) ** 2)
            # if the distance is within the current neighbourhood radius
            if w_dist <= r**2:
                # calculate the degree of influence (based on the 2-D distance)
                influence = calculate_influence(w_dist, r)
                # now update the neuron's weight using the formula:
                # new w = old w + (learning rate * influence * delta)
                # where delta = input vector (t) - old w
                new_w = w + (l * influence * (t - w))
                # commit the new weight
                net[x, y, :] = new_w.reshape(1, 3)
```

In [133]: #show the visualization

```
fig = plt.figure()
# setup axes
ax = fig.add_subplot(111, aspect='equal')
ax.set_xlim((0, net.shape[0]+1))
```

```python
ax.set_ylim((0, net.shape[1]+1))
ax.set_title('Self-Organising Map after %d iterations' % n_iterations)

# The Plot can be seen asa compression of the 3000x3 dataset into a 5x5x3 map
# plot the rectangles
for x in range(1, net.shape[0] + 1):
    for y in range(1, net.shape[1] + 1):
        ax.add_patch(patches.Rectangle((x-0.5, y-0.5), 1, 1,
                     facecolor=net[x-1,y-1,:],
                     edgecolor='none'))
plt.show()
```

4. CNN

I. [2%] Why LeNet-5 is said consist of 5-layer networks? Please explain it!

First Layer:
The input for LeNet-5 is a 32×32 grayscale image which passes through the first convolutional layer with 6 feature maps or filters having size 5×5 and a stride of one. The image dimensions changes from 32x32x1 to 28x28x6.

Second Layer:
Then the LeNet-5 applies average pooling layer or sub-sampling layer with a filter size 2×2 and a stride of two. The resulting image dimensions will be reduced to 14x14x6.

Third Layer:
Next, there is a second convolutional layer with 16 feature maps having size 5×5 and a stride of 1. In this layer, only 10 out of 16 feature maps are connected to 6 feature maps of the previous layer.

The main reason is to break the symmetry in the network and keeps the number of connections within reasonable bounds. That's why the number of training parameters in this layers are 1516 instead of 2400 and similarly, the number of connections are 151600 instead of 240000.

Fourth Layer:
The fourth layer (S4) is again an average pooling layer with filter size 2×2 and a stride of 2. This layer is the same as the second layer (S2) except it has 16 feature maps so the output will be reduced to 5x5x16.

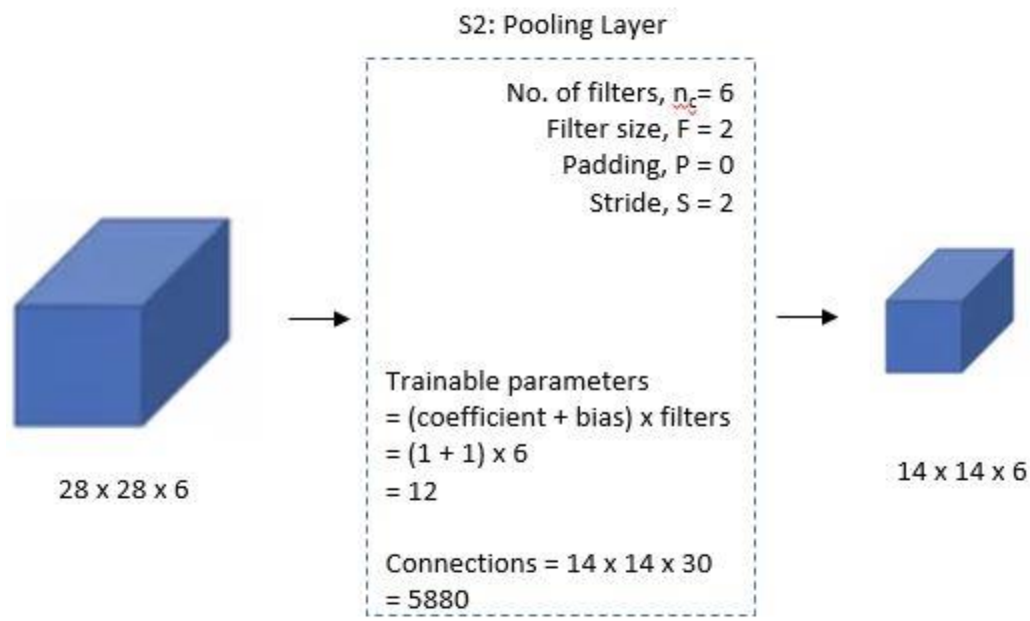
Fifth Layer:
The fifth layer (C5) is a fully connected convolutional layer with 120 feature maps each of size 1×1. Each of the 120 units in C5 is connected to all the 400 nodes (5x5x16) in the fourth layer S4.

Sixth Layer:
The sixth layer is a fully connected layer (F6) with 84 units.

II. [2%] Why in C1 there is 28x28 image pixels? Please sketch the process!

The input for LeNet-5 is a 32×32 grayscale image which passes through the first convolutional layer with 6 feature maps or filters having size 5×5 and a stride of one. The image dimensions changes from 32x32x1 to 28x28x6.

III. [2%] Why in S2 there is 14x14 image pixels? Please draw the process!



IV. [2%] How many numbers of CNN weights in C1 and C3?

C1 weights = 5x5x1x6 = 150

C3 weights = 5x5x6x10 = 1500

C1 + C3 Weights = 1650.

Download Data Set & Normalize

```
import os

os.environ['KERAS_BACKEND'] = 'tensorflow'

from keras.datasets import mnist #28x28

from keras.utils import np_utils

# Load dataset as train and test sets

(x_train, y_train), (x_test, y_test) = mnist.load_data()


# Set numeric type to float32 from uint8

x_train = x_train.astype("float32")

x_test = x_test.astype("float32")

# Normalize value to [0, 1]

x_train /= 255

x_test /= 255

# Transform lables to one-hot encoding

y_train = np_utils.to_categorical(y_train, 10)

y_test = np_utils.to_categorical(y_test, 10)

# Reshape the dataset into 4D array

x_train = x_train.reshape(x_train.shape[0], 28,28,1)

x_test = x_test.reshape(x_test.shape[0], 28,28,1)
```

Define LeNet-5 Model

In [6]:

```
from keras.models import Sequential

from keras import models, layers

import keras

#Instantiate an empty model

model = Sequential()


# C1 Convolutional Layer

model.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation="tanh",
input_shape=(28,28,1), padding="same"))


# S2 Pooling Layer

model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(1, 1), padding="valid"))


# C3 Convolutional Layer

model.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation="tanh",
padding="valid"))


# S4 Pooling Layer

model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding="valid"))


# C5 Fully Connected Convolutional Layer --> matrix

model.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation="tanh",
padding="valid"))
```

#Flatten the CNN output so that we can connect it with fully connected layers

model.add(layers.Flatten())

FC6 Fully Connected Layer --> vectoe

model.add(layers.Dense(84, activation="tanh"))

#Output Layer with softmax activation

model.add(layers.Dense(10, activation="softmax"))

Compile the model

model.compile(loss=keras.losses.categorical_crossentropy, optimizer="SGD", metrics=["accuracy"])

Model training

In [7]:

hist = model.fit(x=x_train,y=y_train, epochs=10, batch_size=128, validation_data=(x_test, y_test), verbose=1)

Out [7]:

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 8s 137us/step - loss: 0.732
3 - accuracy: 0.8054 - val_loss: 0.3543 - val_accuracy: 0.9030
Epoch 2/10
60000/60000 [==============================] - 6s 96us/step - loss: 0.3223
- accuracy: 0.9093 - val_loss: 0.2703 - val_accuracy: 0.9277
Epoch 3/10
60000/60000 [==============================] - 6s 98us/step - loss: 0.2608
- accuracy: 0.9262 - val_loss: 0.2285 - val_accuracy: 0.9372
Epoch 4/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.224
7 - accuracy: 0.9363 - val_loss: 0.2020 - val_accuracy: 0.9442
Epoch 5/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.198
9 - accuracy: 0.9441 - val_loss: 0.1795 - val_accuracy: 0.9503
Epoch 6/10
60000/60000 [==============================] - 6s 99us/step - loss: 0.1783
- accuracy: 0.9493 - val_loss: 0.1628 - val_accuracy: 0.9548
```

```
Epoch 7/10
60000/60000 [==============================] - 6s 99us/step - loss: 0.1611
- accuracy: 0.9547 - val_loss: 0.1473 - val_accuracy: 0.9587
Epoch 8/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.146
3 - accuracy: 0.9593 - val_loss: 0.1356 - val_accuracy: 0.9628
Epoch 9/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.133
9 - accuracy: 0.9621 - val_loss: 0.1237 - val_accuracy: 0.9660
Epoch 10/10
60000/60000 [==============================] - 6s 98us/step - loss: 0.1230
- accuracy: 0.9660 - val_loss: 0.1150 - val_accuracy: 0.9667
```

Evaluate the Model

In [9]:

```
test_score = model.evaluate(x_test, y_test)

print("Test loss {:.4f}, accuracy {:.2f}%".format(test_score[0], test_score[1] * 100))
```

Out [9]:

```
10000/10000 [==============================] - 1s 111us/step
Test loss 0.1150, accuracy 96.67%
```

Visualize the Training Process

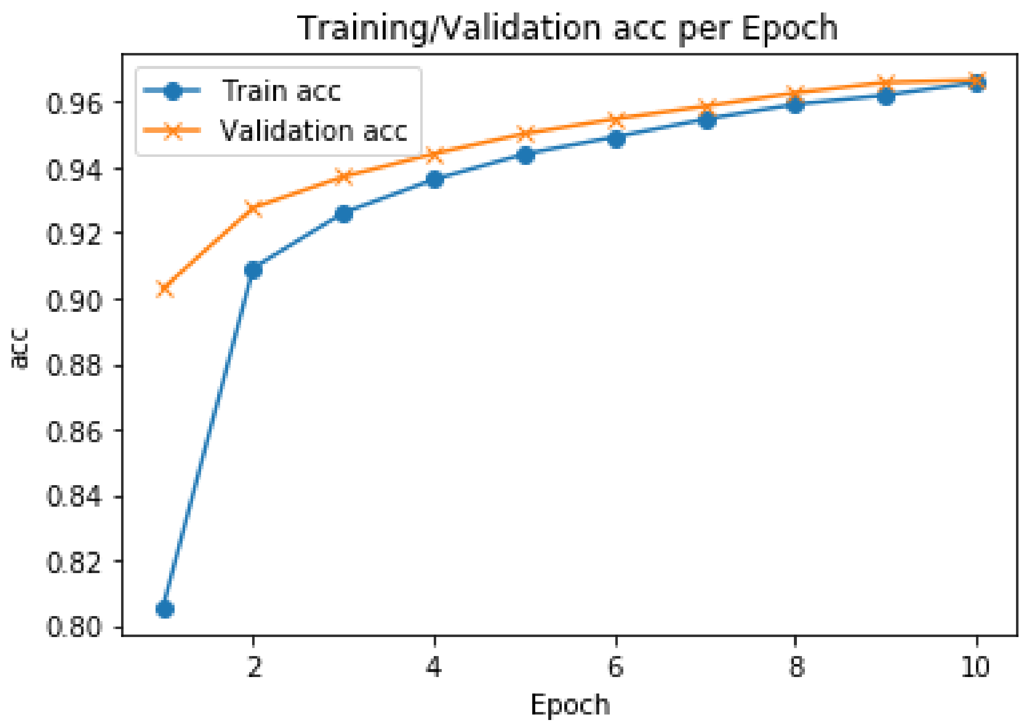
In [17]:

```
import matplotlib.pyplot as plt

f, ax = plt.subplots()

ax.plot([None] + hist.history["accuracy"], "o-")

ax.plot([None] + hist.history["val_accuracy"], "x-")

# Plot legend and use the best location automatically: loc = 0.

ax.legend(["Train acc", "Validation acc"], loc = 0)

ax.set_title("Training/Validation acc per Epoch")

ax.set_xlabel("Epoch")
```

ax.set_ylabel("acc")

Out [17]:

```
Text(0, 0.5, 'acc')
```



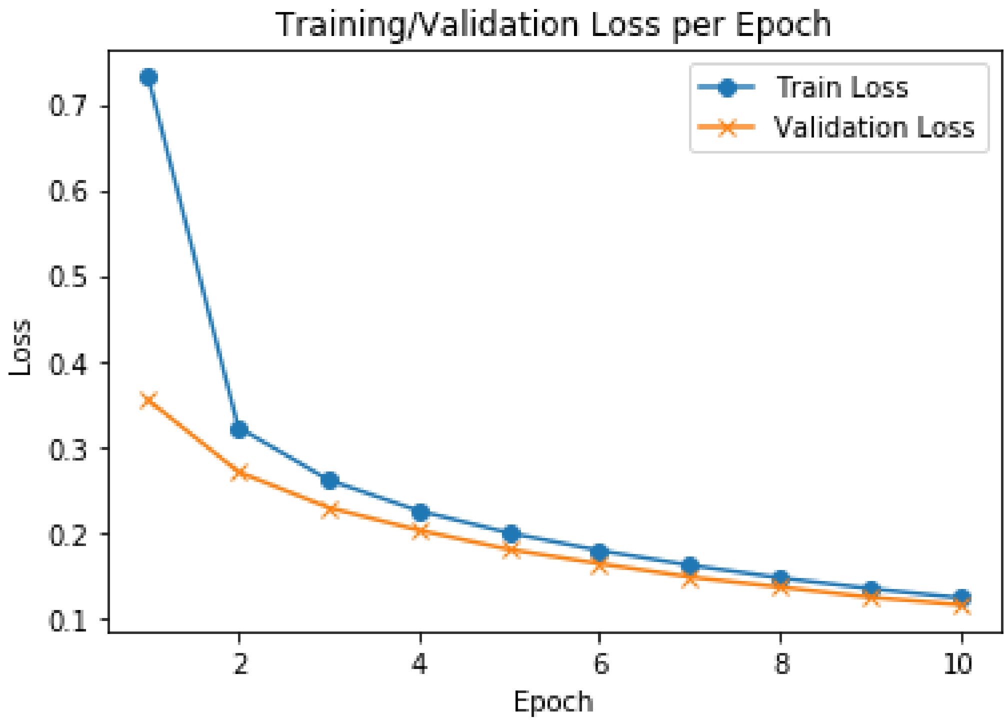
In [18]:

```
import matplotlib.pyplot as plt
f, ax = plt.subplots()
ax.plot([None] + hist.history["loss"], "o-")
ax.plot([None] + hist.history["val_loss"], "x-")
# Plot legend and use the best location automatically: loc = 0.
ax.legend(["Train Loss", "Validation Loss"], loc = 0)
ax.set_title("Training/Validation Loss per Epoch")
```

```
ax.set_xlabel("Epoch")

ax.set_ylabel("Loss")
```

Out [18]:

```
Text(0, 0.5, 'Loss')
```



3. CNN

Load Data

In [1]:
```
import numpy
import skimage.data
from skimage import io
```

In [6]:
```
dataset = []
```
In [7]:
```
dataset = numpy.array([[
  [1, 0, 1, 1, 1, 1, 1, 1, 0, 1],
  [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
  [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 1],
  [1, 0, 1, 0, 0, 0, 0, 1, 1, 1],
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 1],
  [1, 0, 1, 0, 0, 0, 0, 1, 1, 1],
  [1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
  [1, 0, 1, 1, 1, 1, 1, 1, 1, 1]
]])
```

Preparing Filters

In [8]:
```
l1_filter = numpy.zeros((2,3,3))
```
In [9]:
```
l1_filter[0, :, :] = numpy.array([[[0, 0.25, 0],
                                   [0.25, 0.25, 0.25],
                                   [0, 0.25, 0]]])
```
In [10]:
```
l1_filter[1, :, :] = numpy.array([[[1,   0,  -1],
                                   [1,   0,  -1],
                                   [1, 0, -1]]])
l1_filter.shape
```

Convolution

In [11]:

```
def conv(dataset, conv_filter):

    if len(dataset.shape) != len(conv_filter.shape) - 1: # Check whether number
of dimensions is the same
        print("Error: Number of dimensions in conv filter and image do not
match.")
        exit()
    if len(dataset.shape) > 2 or len(conv_filter.shape) > 3: # Check if number of
image channels matches the filter depth.
        if dataset.shape[-1] != conv_filter.shape[-1]:
            print("Error: Number of channels in both image and filter must
match.")

    if conv_filter.shape[1] != conv_filter.shape[2]: # Check if filter dimensions
are equal.
        print('Error: Filter must be a square matrix. I.e. number of rows and
columns must match.')

    if conv_filter.shape[1]%2==0: # Check if filter diemnsions are odd.
        print('Error: Filter must have an odd size. I.e. number of rows and
columns must be odd.')


    # An empty feature map to hold the output of convolving the filter(s) with
the dataset.
    feature_maps = numpy.zeros((dataset.shape[0]-conv_filter.shape[1]+1,
                                dataset.shape[1]-conv_filter.shape[1]+1,
                                conv_filter.shape[0]))

    # Convolving the dataset by the filter(s).
    for filter_num in range(conv_filter.shape[0]):
        print("Filter ", filter_num + 1)
        curr_filter = conv_filter[filter_num, :] # getting a filter from the
bank.
        """
        Checking if there are mutliple channels for the single filter.
        If so, then each channel will convolve the image.
        The result of all convolutions are summed to return a single feature map.
        """
        if len(curr_filter.shape) > 2:
```

```
            conv_map = conv_(dataset[:, :, 0], curr_filter[:, :, 0]) # Array
holding the sum of all feature maps.
            for ch_num in range(1, curr_filter.shape[-1]): # Convolving each
channel with the image and summing the results.
                conv_map = conv_map + conv_(dataset[:, :, ch_num],
                                  curr_filter[:, :, ch_num])
        else: # There is just a single channel in the filter.
            conv_map = conv_(dataset, curr_filter)
        feature_maps[:, :, filter_num] = conv_map # Holding feature map with the
current filter.
    return feature_maps # Returning all feature maps.


def conv_(dataset, conv_filter):
    filter_size = conv_filter.shape[1]
    result = numpy.zeros((dataset.shape))
    #Looping through the image to apply the convolution operation.
    for r in numpy.uint16(numpy.arange(filter_size/2.0,
                          dataset.shape[0]-filter_size/2.0+1)):
        for c in numpy.uint16(numpy.arange(filter_size/2.0,
                                           dataset.shape[1]-filter_size/2.0+1)):
            """
            Getting the current region to get multiplied with the filter.
            How to loop through the dataset and get the region based on
            the dataset and filer sizes is the most tricky part of convolution.
            """
            curr_region = dataset[r-
numpy.uint16(numpy.floor(filter_size/2.0)):r+numpy.uint16(numpy.ceil(filter_size/
2.0)),
                              c-
numpy.uint16(numpy.floor(filter_size/2.0)):c+numpy.uint16(numpy.ceil(filter_size/
2.0))]
            #Element-wise multipliplication between the current region and the
filter.
            curr_result = curr_region * conv_filter
            conv_sum = numpy.sum(curr_result) #Summing the result of
multiplication.
            result[r, c] = conv_sum #Saving the summation in the convolution
layer feature map.

    #Clipping the outliers of the result matrix.
    final_result = result[numpy.uint16(filter_size/2.0):result.shape[0]-
numpy.uint16(filter_size/2.0),
                          numpy.uint16(filter_size/2.0):result.shape[1]-
numpy.uint16(filter_size/2.0)]
```

```
    return final_result
```

In [12]:

```
l1_feature_map = conv(dataset, l1_filter)
l1_feature_map.shape
```

In [13]:

```
for i in range(2):
    dataset = l1_feature_map[:,:,i]
    io.imshow(dataset)
    io.show()
```

Relu Activation Function

In [14]:

```
def relu(feature_map):
    #Preparing the output of the ReLU activation function.
    relu_out = numpy.zeros(feature_map.shape)
    for map_num in range(feature_map.shape[-1]):
        for r in numpy.arange(0,feature_map.shape[0]):
            for c in numpy.arange(0, feature_map.shape[1]):
                relu_out[r, c, map_num] = numpy.max([feature_map[r, c, map_num],
0])
    return relu_out
```

In [15]:

```
l1_feature_map_relu = relu(l1_feature_map)
l1_feature_map_relu.shape

for i in range(2):
    dataset = l1_feature_map_relu[:,:,i]
    io.imshow(dataset)
    io.show()
```

Max Pooling Step

In [16]:
```
def pooling(feature_map, size=2, stride=2):
    #Preparing the output of the pooling operation.
    pool_out = numpy.zeros((numpy.uint16((feature_map.shape[0]-size+1)/stride+1),
                            numpy.uint16((feature_map.shape[1]-size+1)/stride+1),
                            feature_map.shape[-1]))
    for map_num in range(feature_map.shape[-1]):
        r2 = 0
        for r in numpy.arange(0,feature_map.shape[0]-size+1, stride):
            c2 = 0
            for c in numpy.arange(0, feature_map.shape[1]-size+1, stride):
                pool_out[r2, c2, map_num] = numpy.max([feature_map[r:r+size,
c:c+size, map_num]])
                c2 = c2 + 1
            r2 = r2 +1
    return pool_out
```

In [17]:
```
l1_feature_map_relu_pool = pooling(l1_feature_map_relu, 2, 2)
l1_feature_map_relu_pool.shape
```

In [18]:
```
for i in range(2):
    dataset = l1_feature_map_relu_pool[:,:,i]
    io.imshow(dataset)
    io.show()
```

Stacking Layers

In [19]:
```
# Second conv layer
l2_filter = numpy.random.rand(3, 5, 5, l1_feature_map_relu_pool.shape[-1])
print("\n**Working with conv layer 2**")
l2_feature_map = conv(l1_feature_map_relu_pool, l2_filter)
print("\n**ReLU**")
l2_feature_map_relu = relu(l2_feature_map)
print("\n**Pooling**")
l2_feature_map_relu_pool = pooling(l2_feature_map_relu, 2, 2)
```

```python
print("**End of conv layer 2**\n")
```

In [20]:

```python
for i in range(3):
    dataset = l2_feature_map_relu_pool[:,:,i]
    io.imshow(dataset)
    io.show()
```

In [21]:

```python
# Third conv layer
l3_filter = numpy.random.rand(1, 7, 7, l2_feature_map_relu_pool.shape[-1])
print("\n**Working with conv layer 3**")
l3_feature_map = conv(l2_feature_map_relu_pool, l3_filter)
print("\n**ReLU**")
l3_feature_map_relu = relu(l3_feature_map)
print("\n**Pooling**")
l3_feature_map_relu_pool = pooling(l3_feature_map_relu, 2, 2)
print("**End of conv layer 3**\n")
```

In [22]:

```python
for i in range(1):
    dataset = l3_feature_map_relu_pool[:,:,i]
    io.imshow(dataset)
    io.show()
```

Notes: Terimakasih kepada pak Agung yg telah mengajarkan saya matakuliah Artificial Neural Network. Saya beruntung dapat dosen seperti bapak yang mengajarkan materi ini dengan sangat jelas dan baik.