Hopfield_1D.py
As adapted from Introduction to Computer Simulation Methods by Gould and Tobochnik

Evan Breitsch

April 18, 2014

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Feb  9 22:12:40 2014
4  # Program hopfield
5  # This program is adopted from the algorithm provided
6  # for a 1 dimensional Hopfield network in Neural Networks, Introduction to
       computer
7  # programming - Gould (pg 527-530)
8  @author: Evan Breitsch
9  """
10
11 import numpy as np
12 import os
13 import time
14
15 maxNumpat = 2            # enter the max number of patterns to store
16 maxPatlen = 6            # enter the max length of the patterns
17 numberOfTrials = 1       # enter how many times to iterrate each network
18
19 def main():
20     t0 = time.time()
21     avgHamDistances = np.zeros((maxNumpat,maxPatlen))
22     avgErrors = np.zeros((maxNumpat,maxPatlen))
23     avgRecallTimes = np.zeros((maxNumpat,maxPatlen))
24     avgMemorizeTimes = np.zeros((maxNumpat,maxPatlen))
25     avgTotalTimes = np.zeros((maxNumpat,maxPatlen))
26     seedForRand = 997 #1234
27     for numpat in range(1,maxNumpat +1): #iterates the number of patterns
28         print numpat, time.time()
29         for patlen in range(1, maxPatlen +1): #iterates pattern length
30             hams = []
31             times = []
32             for trial in range(numberOfTrials): #iterates the trials for each
33                                                 #network of pattern size and #
```

```
34
35                  randomObj = np.random.RandomState(seedForRand)
36                  seedForRand += 1
37                  patList = randomObj.randint(0,2, size=(numpat, patlen))
38                  randomInt = randomObj.randint(0, numpat, 1)
39                  targetPat = patList[randomInt][:]
40
41                  tm1 = time.time()
42                  memory = memorize(patList, numpat, patlen)
43                  tm2 = time.time()
44                  memorizeTime = tm2 - tm1
45
46                  tr1 = time.time()
47                  hammingDistance = recall(memory, targetPat)
48                  tr2 = time.time()
49                  recallTime = tr2 - tr1
50
51                  hams.append(hammingDistance)
52                  times.append(memorizeTime + recallTime)
53
54              avgTime = float(sum(times))/numberOfTrials
55              avgDis = float(sum(hams))/numberOfTrials
56              avgError = avgDis/patlen
57              avgHamDistances[numpat-1][patlen-1] = avgDis
58              avgErrors[numpat-1][patlen-1] = avgError
59              avgRecallTimes[numpat-1][patlen-1] = recallTime
60              avgMemorizeTimes[numpat-1][patlen-1] = memorizeTime
61              avgTotalTimes[numpat-1][patlen-1] = avgTime
62              avgErrors[numpat-1][patlen-1] = avgError
63
64 #
65      saveHamDistances(avgHamDistances)
66  #    showPlot(avgRecallTimes, 'Recall Times')
67      showPlot(avgHamDistances, 'Hamming Distance')
68  #    showPlot(avgTotalTimes, 'Total Times')
69 #     showPlot(avgMemorizeTimes, "Memorize Times")
70      showPlot(avgErrors, "Hamming Error (Distance Normalized)")
71      t1 = time.time()
72      timeTaken = t1 - t0
73
74      print "-----------------------------------"
75      print "Max # patterns:        ", maxNumpat
76      print "Max length of pattern: ", maxPatlen
77      print "Number of Trials:      ", numberOfTrials
78      print "Time Taken:            ", timeTaken
79      print "Memorize time          ", memorizeTime
80      print "Recall time            ", recallTime
81      print "end of run"
82
83
84 def memorize(patList, numpat, patlen): # convert values from binary to bipolar
```

```
85                                          # find kronecker product, and set
86                                          # the diagonal to zeros
87      memory = np.zeros((patlen, patlen), dtype=int)
88      for pat in patList:
89
90          zeros = pat == 0
91          pat = pat - zeros
92          tpat = np.array([pat]).T
93          relation = np.kron(pat, tpat)
94          np.fill_diagonal(relation,0)
95          memory = memory + relation
96          print memory
97
98      return memory
99
100
101 def recall(memory, targetPat): #perform matrix multiplication then
102                                 # convert result to binary output
103                                 # compare output to targetPat to
104                                 # find the Hamming distance
105
106     output = np.multiply(memory, targetPat)
107     output = np.sum(output, axis=1)
108     negative_indices = output < 0
109     output[negative_indices] = 0
110     positive_indices = output > 0
111     output[positive_indices] = 1
112     misMatch = np.not_equal(output, targetPat)
113     hammingDistance = np.sum(misMatch)
114     return hammingDistance
115
116
117 def saveHamDistances(avgHamDis): # saves the data into a CSV file
118
119     hamFile = ('hammingDistance_Len-'+ str(maxPatlen) + "_Num-"
120                 + str(maxNumpat) + '_Trials-'+ str(numberOfTrials) + '.csv')
121     try:
122         (open(hamFile))
123     except IOError:
124         f = open(hamFile, 'a')
125     else:
126         os.remove(hamFile)
127         f = open(hamFile, 'a')
128     for row in avgHamDis:
129         for col in row:
130             f.write(str(col) +',')
131         f.write('\n')
132     f.close()
133
134 def showPlot(data, title):  # makes a matplotlib surface plot of data
135     import matplotlib.pyplot as plt
```

```
136     from mpl_toolkits.mplot3d import Axes3D # this is actually needed
137
138     fig = plt.figure()
139     ax = fig.gca(projection='3d')
140     X = np.arange(1, maxPatlen + 1, 1)
141     Y = np.arange(1, maxNumpat + 1, 1)
142     X, Y = np.meshgrid(X, Y)
143     Z = data
144
145     ax.plot_surface(X, Y, Z, rstride=1, cstride=1, alpha =0.3, cmap="hot")
146
147     ax.set_xlabel('Pattern Length')          # axis labels
148     ax.set_xlim(0, maxPatlen)
149     ax.set_ylabel('Number of Patterns')
150     ax.set_ylim(0, maxNumpat + 2)
151     ax.set_zlabel(title)
152     ax.set_zlim(0, np.amax(Z))
153
154     plt.show()
155
156 if (__name__ == "__main__"):
157     main()
```