Bayesian Derived Hopfield Networks

Akiva Lipshitz

We derived the hopfield network in the paper *Deriving a Hopfield Network from The Bayesian Perspective* (Lipshitz 2017).

This program is a demonstration that the hopfield network does in fact work as expected.

The governing update equation is:

$$z_j := \tanh\left\{\sum_{\mu=1}^{N} x_j^\mu \alpha^{H_\mu}\right\}$$

where

$$H_\mu = \frac{1}{2}\left[N - \sum_{j=1}^{T} z_j x_j^\mu\right]$$

To avoid numerical overflow, we will use a first order taylor series to approximate the exponential in the update equation. Because we only care about sign and not magnitude, this is a fine approximation.

$$z_j := \tanh\left\{\sum_{\mu=1}^{N} x_j^\mu H_\mu \ln \alpha\right\}$$

```
from pylab import *
from __future__ import division
import imageio as imio
import glob
```

```
p = 0.2
T = 100
N = 25
```

```
def noise(p):
    """"
    p is the probability of switching
    """
    zeta = np.random.binomial(1, p, size=T*T).reshape((T,T));
    nzeta = zeta.copy();
    nzeta[zeta==1] = -1;
    nzeta[zeta==0] = 1;
    return nzeta
```

```
def squash(a1,b1, a2,b2, x):
    return a1 + (b1-a1)*(x/(b2-a2))

def randpol(D=T):
    a = 1;
    sigmas = tile(np.linspace(0,1,num=N), (2,1))
    mus = tile(np.random.uniform(-1,1,size=N), (2,1))
    p = np.random.normal(
            scale=sigmas.flatten(),
            loc=mus.flatten(),
            size=2*N).reshape((2,N))
    xs = linspace(-a, a, num=D)
    ys = linspace(-a,a, num=D)
    zsx = polyval(p[0,:], xs)
    zsy = polyval(p[1,:], ys)
    grid = meshgrid(zsx, zsy)
    grid = np.array(grid)
    zs = sum(grid, axis=0)
    return zs

def randBinPol(*args, **kwargs):
    """
    This is just a scheme for generating
    correlated training
    """
    zs = randpol(*args, **kwargs)
    z = zs.copy()
    z[zs<=mean(zs)] = -1.
    z[zs>mean(zs)] = 1.
    return z

lib = np.zeros((N,T,T))
for i in range(N):
    zs = randBinPol(D=T)
    lib[i,:,:] = zs

dims = int(round(sqrt(N)))
figure(figsize=(dims*2, dims*2));
a = 1
num = N
for i in range(num):
    zeta = noise(p);
    subplot(dims,dims,a);
    imshow(lib[i]);
    title("Image {}".format(a))
    a+=1
tight_layout()
```

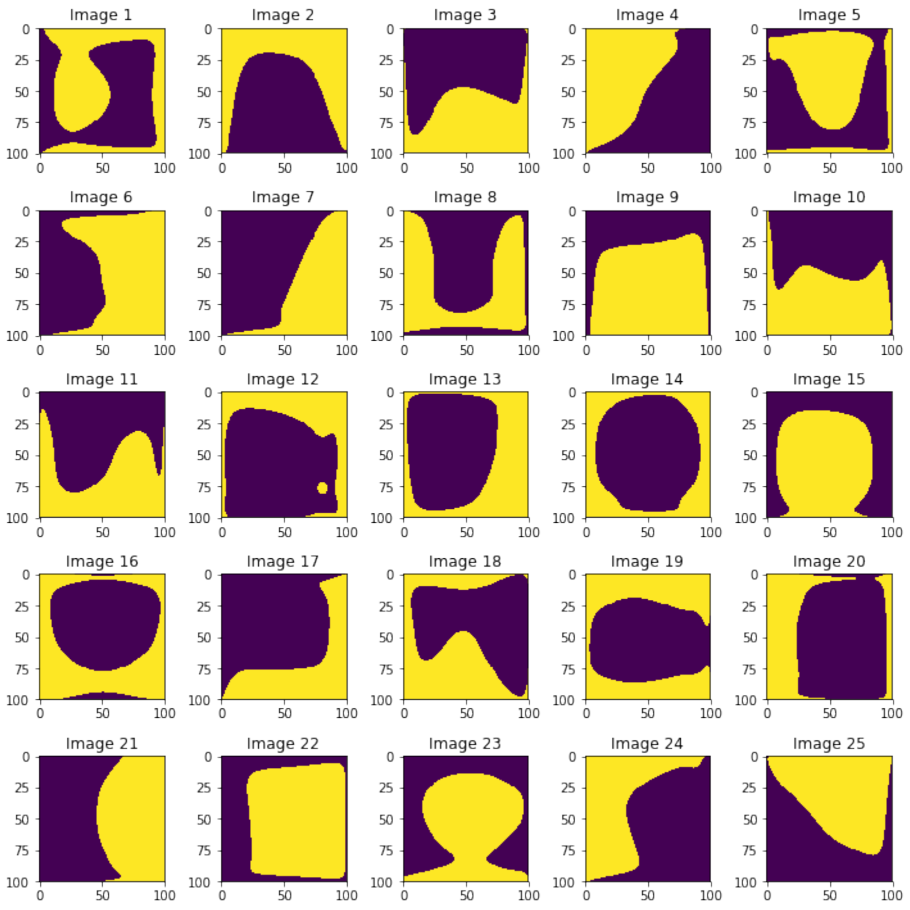

Figure 1: png

```
lib = lib.reshape((N,T*T))

class Hopnet:
    def __init__(self, p, xs):
        """
        Initializes a hopfield network.
        Assumes the distribution over observed
        signals is already known.

        We will work on a gibbs sampler for real
        time bayesian updates to the
        signal noise
        distribution.
        """
        N, T = xs.shape[0], xs.shape[1]
        self.z = zeros(T)#.astype(np.float64);
        self.xs = xs#.astype(np.float64);
        self.N = np.array(N)#.astype(np.float64);
        self.T = np.array(T)#.astype(np.float64);
        self.alpha = np.array(p/(1-p))#.astype(np.float64);

    def initialize(self, z0):
        self.z = z0.astype(np.float64)
        return self

    def H(self):
        """
        Returns a 1*N matrix of hamming distances
        """
        r= 0.5*(self.N-self.z.dot(self.xs.T));
        return r


    def propogate(self):
        inner =  self.xs.T*np.log(self.alpha)*self.H();
        self.z = sign(sum(inner, axis=1));
        return self.z


hop = Hopnet(p, lib)

niter = 10

figure(figsize=(10,2*niter))
a=1
hop.initialize(lib[np.random.random_integers(N-1)]*noise(p).flatten())
subplot(niter,2,1)
imshow(hop.z.reshape(T,T))
title("HopNet Iteration {}".format(0))
colorbar()
for i in range(niter-1):
    a+=1
    z = hop.propogate().reshape((T,T))
    subplot(niter,2,a)
    imshow(z)
    colorbar()
    title("HopNet Iteration {}".format(i+1))
tight_layout()
suptitle("Iterations for Hopfield Network", y=1.08, fontdict={"fontsize":15})
```

```
/Users/akivalipshitz/anaconda/lib/python2.7/site-packages/ipykernel_launcher.py:5: Deprecati
  """
```

```
<matplotlib.text.Text at 0x13c19af90>
```

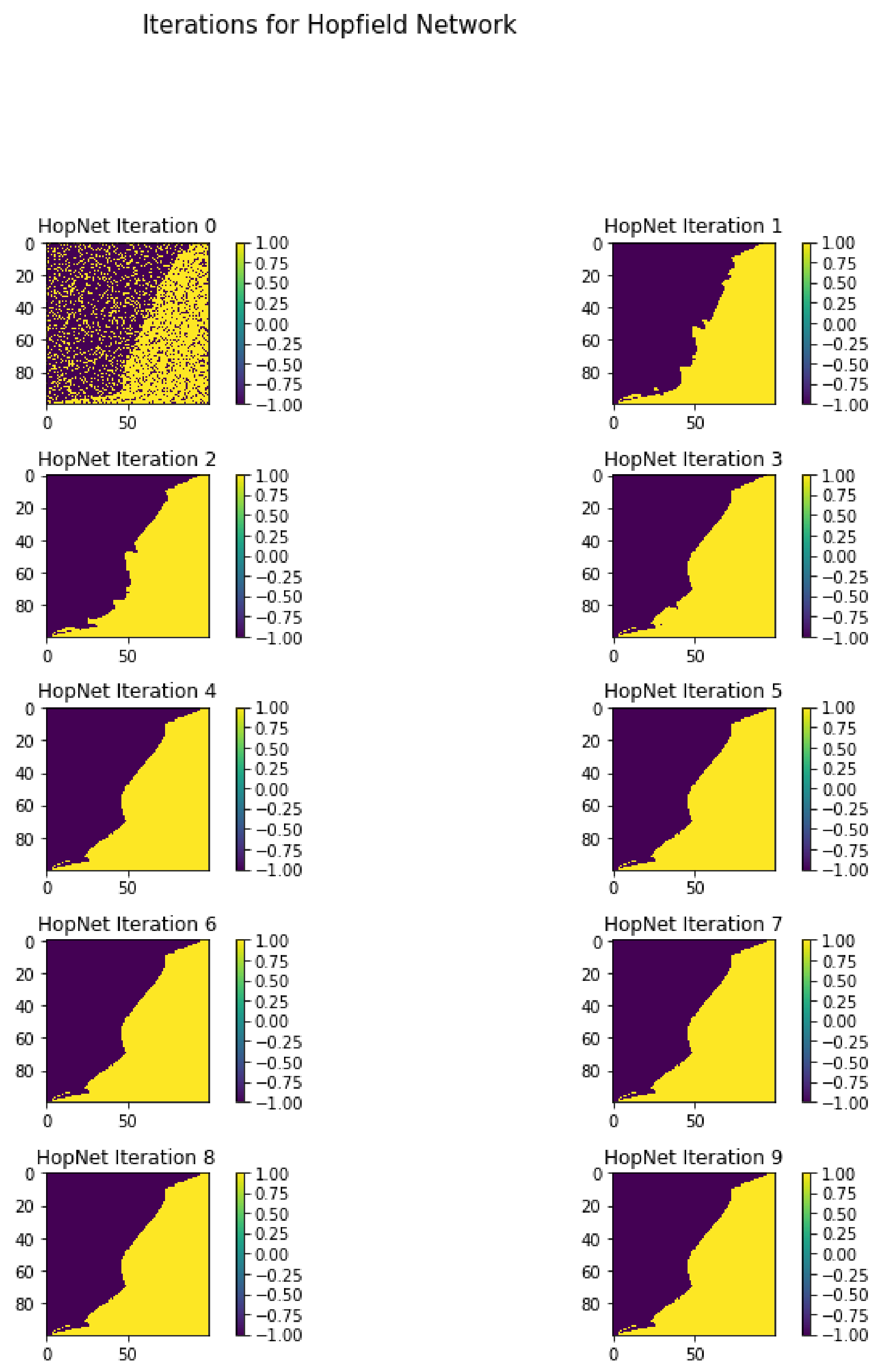
Iterations for Hopfield Network
HopNet Iteration 0
HopNet Iteration 1
HopNet Iteration 2
HopNet Iteration 3
HopNet Iteration 4
HopNet Iteration 5
HopNet Iteration 6
HopNet Iteration 7
HopNet Iteration 8
HopNet Iteration 9
0
20
40
60
80
0
50
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Figure 2: png