simplest-ever-multicause-RBM

June 27, 2015

1 2 visibles, 2 rbms, with 2 hiddens each, and the same weights!

I'm making a super simple example network here, to reality-check our thinking and test the thing out in the smallest possible example.

In [1]:
```
%matplotlib inline
import numpy as np
import numpy.random as rng
from pylab import *

def sigmoid(x):
    return 1.0/(1+np.exp(-x))
```

In [2]:
```
# This is showing the action in hA when v is clamped on visible,
# and hB is the pattern on the hidden units of the OTHER rbm.
def do_APPROX_figure(fig_name, w, v, hB):
    hB = hB.reshape((2,1))
    pats = np.array([[0,0],[0,1],[1,0],[1,1]])
    hA_prob = np.zeros(pats.shape, dtype=float)
    for row, hA in enumerate(pats):
        hA = hA.reshape((2,1))
        phiA, phiB = np.dot(w,hA), np.dot(w,hB)
        phiA_alt = phiA - hA*w + 0.5*w
        phiB_alt = phiB - hB*w + 0.5*w
        # i.e. phi_alts should now be same shape as w in fact! :(
        sigA_to_A  = sigmoid(phiA_alt)
        sigAB_to_A  = sigmoid(phiA_alt + phiB)
        effective_visA = v + sigA_to_A - sigAB_to_A
        our_psiA = (effective_visA * w).sum(1)
        hA_prob[row,:] = sigmoid(our_psiA)

    #I DON'T BELIEVE THE FOLLOWING LINE WAS RIGHT AFTERALL.....
    #phiA = np.dot(pats, w) - np.dot((2*pats-1),w/2)
    #phiB = np.dot(hB, w)
    #sigA  = sigmoid(phiA)
    #sigAB = sigmoid(phiA + phiB)
    #effective_vis = v + sigA - sigAB
    #psiA = np.dot(v, w)
    #our_psiA = np.dot(effective_vis, w)

    subplot(259)
    imshow(v, interpolation='nearest',cmap='copper', vmin=0, vmax=1)
    title('visible')
```

```
        ax = axis('off')
        subplot(255)
        imshow(hB.T, interpolation='nearest',cmap='copper', vmin=0, vmax=1)
        title('rbmB')
        ax = axis('off')
        subplot(251)
        imshow(pats, interpolation='nearest',cmap='copper', vmin=0, vmax=1)
        title('rbmA')
        ax = axis('off')
        subplot(252)
        imshow(hA_prob, interpolation='nearest',cmap='copper', vmin=0, vmax=1)
        title('orbm Pr(A)')
        ax = axis('off')
        subplot(253)
        psiA = np.dot(v, w)
        imshow(sigmoid(psiA), interpolation='nearest',cmap='copper', vmin=0, vmax=1)
        title('rbm Pr(A)')
        #colorbar()
        ax = axis('off')
        savefig(fig_name)
```

```
In [3]: # This is showing the action in hA when v is clamped on visible,
        # and hB is the pattern on the hidden units of the OTHER rbm.
        def do_EXACT_figure(fig_name, w, v, hB):
            hB = hB.reshape((2,1))
            phiB = np.dot(w,hB)
            pats = np.array([[0,0],[0,1],[1,0],[1,1]])
            hA_prob = np.zeros(pats.shape, dtype=float)
            for row, hA in enumerate(pats):
                hA = hA.reshape((2,1))
                phiA = np.dot(w,hA)
                phiA0 = phiA - hA*w # phiA vector (cols) when each hidden in turn (rows) is off
                C = np.log(sigmoid(phiA0))
                C = C - np.log(sigmoid(phiA0 + w))
                C = C + np.log(sigmoid(phiA0 + w + phiB))
                C = C - np.log(sigmoid(phiA0 + phiB))

                our_psiA = (v*w + C).sum(1)
                hA_prob[row,:] = sigmoid(our_psiA)

            #I DON'T BELIEVE THE FOLLOWING LINE WAS RIGHT AFTERALL.....
            #phiA = np.dot(pats, w) - np.dot((2*pats-1),w/2)
            #phiB = np.dot(hB, w)
            #sigA  = sigmoid(phiA)
            #sigAB = sigmoid(phiA + phiB)
            #effective_vis = v + sigA - sigAB
            #psiA = np.dot(v, w)
            #our_psiA = np.dot(effective_vis, w)

            subplot(259)
            imshow(v, interpolation='nearest',cmap='copper', vmin=0, vmax=1)
            title('visible')
            ax = axis('off')
            subplot(255)
```

```
imshow(hB.T, interpolation='nearest',cmap='copper', vmin=0, vmax=1)
title('rbmB')
ax = axis('off')
subplot(251)
imshow(pats, interpolation='nearest',cmap='copper', vmin=0, vmax=1)
title('rbmA')
ax = axis('off')
subplot(252)
imshow(hA_prob, interpolation='nearest',cmap='copper', vmin=0, vmax=1)
title('orbm Pr(A)')
ax = axis('off')
subplot(253)
psiA = np.dot(v, w)
imshow(sigmoid(psiA), interpolation='nearest',cmap='copper', vmin=0, vmax=1)
title('rbm Pr(A)')
#colorbar()
ax = axis('off')
savefig(fig_name)
```

I will set weights such that patterns 01 and 10 on the visible units get piles of sand (are "memories"), and 00 and 11 aren't.

```
In [4]: w = 1.0 * np.array([[1,-1],[-1,1]])
```

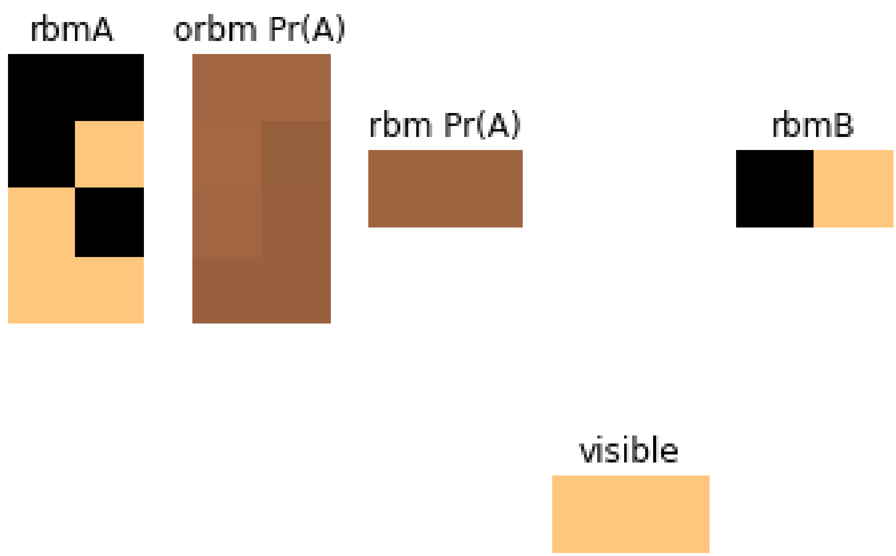
1.0.1 explaining away, example 1

We clamp a visible pattern that requires both the rbms to pitch in. The B network explains the right-most bit.

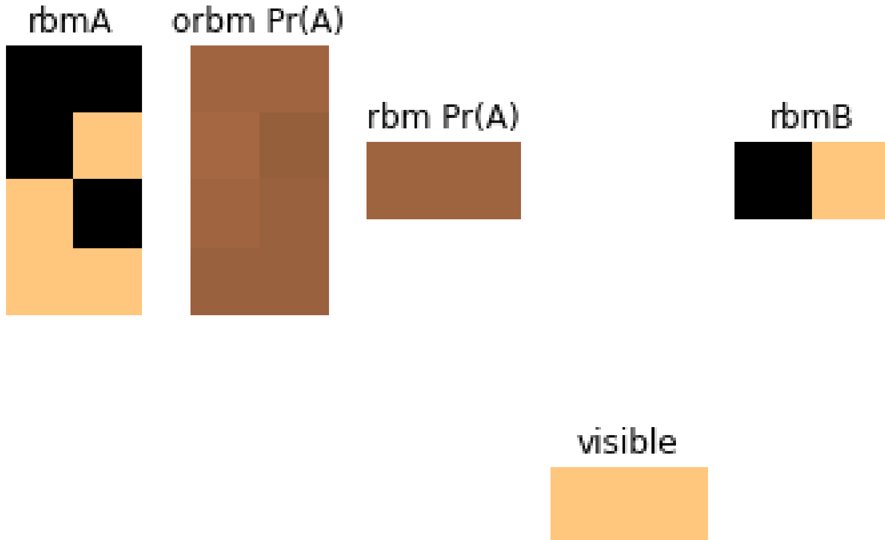
Under regular rbm dynamics, the A network is ambivalent as to which hidden state to choose: all 4 are equally likely.

But under orbm dynamics we would expect the A network to be asked to explain the left-most bit.

```
In [5]: do_APPROX_figure('the_way', w, v=np.array([[1,1]]), hB = np.array([[0,1]]))
```



```
In [6]: do_EXACT_figure('the_way', w, v=np.array([[1,1]]), hB = np.array([[0,1]]))
```



Fuckit. This seemed to be making sense Friday. But I think we'd done the calculation wrong. However this still doesn't match the hand-calculated one, which DID seem sensible.

What's alarming is that both calculations (exact and approx) seem to be agreeing on a shit answer.

Note: we show the rbmA state alongside the orbm's Pr(A) here because in theory the orbm probabilities do depend on the existing state. However in this particular example they don't - perhaps because it's just so symmetrical.

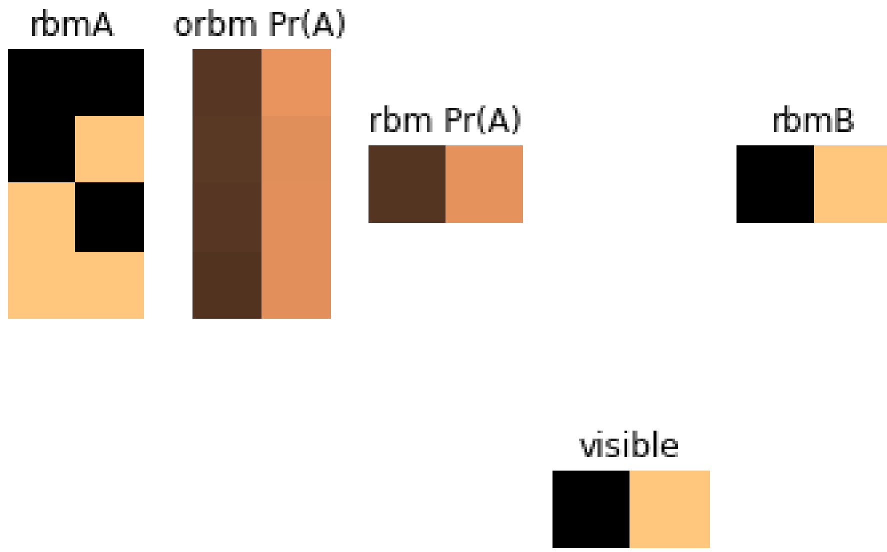
1.0.2 explaining away, example 2

Suppose the entire visible pattern is nicely explained by the B network. What does this mean for the A one?

The standard rbm would try to account for the image.

In the orbm, here's what happens:

```
In [8]: do_EXACT_figure('the_way', w, v=np.array([[0,1]]), hB = np.array([[0,1]]))
```



It's still trying, and just as hard...

2 Learning

In [143]:
```
# Our convention: hiddens are rows, visibles are columns.
# So our weights are W[hid,vis].
# Our hidden states should always be column vectors.
# Our visible states should always be row vectors.
# psi always refers to hiddens, phi always to visibles: WATCH OUT FUKKAHS.

w = 1.0 * np.array([[1,-1],[-1,1]])

vpats  = np.array([[0,1],[1,0]])# our data points
eta = .05 # learning rate
print('initial weights: \n', w)
for step in range(300):
    # Wake phase
    for v in vpats: # 'clamped'
        v = v.reshape(1,2)
        #print('shape of v should be 1,2: ',v.shape)
        # SAMPLE from hA and hB
        # Start by initialising as if it were a vanilla RBM
        psiA, psiB = np.dot(v,w), np.dot(v,w)
        hA_prob, hB_prob = sigmoid(psiA), sigmoid(psiB)
        hA = (hA_prob > rng.random(size=hA_prob.shape)).reshape(2,1)
        hB = (hB_prob > rng.random(size=hB_prob.shape)).reshape(2,1)
        #print('shape of hA should be 2,1: ',hA.shape)
        for t in range(10):
            phiA, phiB = np.dot(w,hA), np.dot(w,hB)
            phiA_alt = phiA - hA*w + 0.5*w
            phiB_alt = phiB - hB*w + 0.5*w
```

```
                # i.e. phi_alts should now be same shape as w in fact! :(
                if (step==1) and (t==1): print('phiA_alt shape is ',phiA_alt.shape)
                sigA_to_A  = sigmoid(phiA_alt)
                sigB_to_B  = sigmoid(phiB_alt)
                sigAB_to_A  = sigmoid(phiA_alt + phiB)
                sigAB_to_B  = sigmoid(phiB_alt + phiA)

                effective_visA = v + sigA_to_A - sigAB_to_A
                effective_visB = v + sigB_to_B - sigAB_to_B
                our_psiA = (effective_visA * w).sum(1)
                our_psiB = (effective_visB * w).sum(1)
                hA_prob = sigmoid(our_psiA)
                hB_prob = sigmoid(our_psiB)
                hA = (hA_prob > rng.random(size=hA_prob.shape)).reshape(2,1)
                hB = (hB_prob > rng.random(size=hB_prob.shape)).reshape(2,1)

            dwA = sigmoid(phiA_norm)*hA + (v - sigAB_toA)*hA
            dwB = sigmoid(phiB_norm)*hB + (v - sigAB_toB)*hB
            #COMBINE
            w = w + eta * (dwA + dwB)

        # Sleep phase
        for v in vpats: # 'clamped'
            for t in range(10):
                # visibles to hiddens
                psiA, psiB = np.dot(v,w), np.dot(v,w)
                hA_prob, hB_prob = sigmoid(psiA), sigmoid(psiB)
                hA = (hA_prob > rng.random(size=hA_prob.shape)).reshape(2,1)
                hB = (hB_prob > rng.random(size=hB_prob.shape)).reshape(2,1)
                # hiddens to visibles
                phiA, phiB = np.dot(w,hA), np.dot(w,hB)
                vA_prob, vB_prob = sigmoid(phiA), sigmoid(phiB)
                vA = (vA_prob > rng.random(size=vA_prob.shape)).reshape(1,2)
                vB = (vB_prob > rng.random(size=vB_prob.shape)).reshape(1,2)

            dwA = visA*hA
            dwB = visB*hB
            #COMBINE
            #w = w - eta * (dwA + dwB)

    print('new weights: \n',w)
```

```
initial weights:
 [[ 1. -1.]
 [-1.  1.]]
phiA_alt shape is  (2, 2)
phiA_alt shape is  (2, 2)
new weights:
 [[ 20.73028589  35.24945651]
 [ 25.48169886  27.19015796]]
```

In [144]:
```
visAprob = sigmoid(np.dot(hA,w))
visA = 1.0*(visAprob > rng.random(size=(1,2)))
psiAprob = sigmoid(np.dot(visA, w))
hA = 1.0*(psiAprob > rng.random(size=(1,2)))
```

```
print(visA)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)

    <ipython-input-144-91af2112cb29> in <module>()
----> 1 visAprob = sigmoid(np.dot(hA,w))
      2 visA = 1.0*(visAprob > rng.random(size=(1,2)))
      3 psiAprob = sigmoid(np.dot(visA, w))
      4 hA = 1.0*(psiAprob > rng.random(size=(1,2)))
      5 print(visA)


    ValueError: objects are not aligned
```

2.0.3 TODO:

1. two-bit learning
2. 3-visible-bit RBMs, two of them with shared weights.

   Something like * 001 has logP=1 * 011 has logP=2 * 111 has logP=1

3. same but with two sets of weights, learning both of them.

   Where the complement is * 110 has logP=1 * 100 has logP=2 * 000 has logP=1

```
In []: z = (rng.random(size=(2,3)).sum(1) > rng.random(size=(2,))).reshape(2,1)
       print (z.shape)
```

```
In []:
```

```
In []:
```