

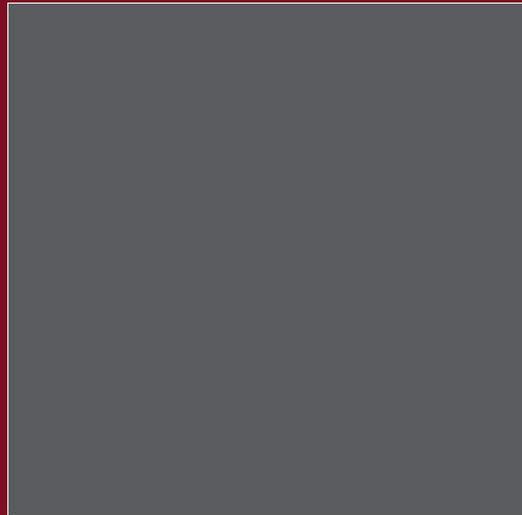
Pattern Recognition with Deep Boltzman Machines

Natasha Parikh and Miranda Parker

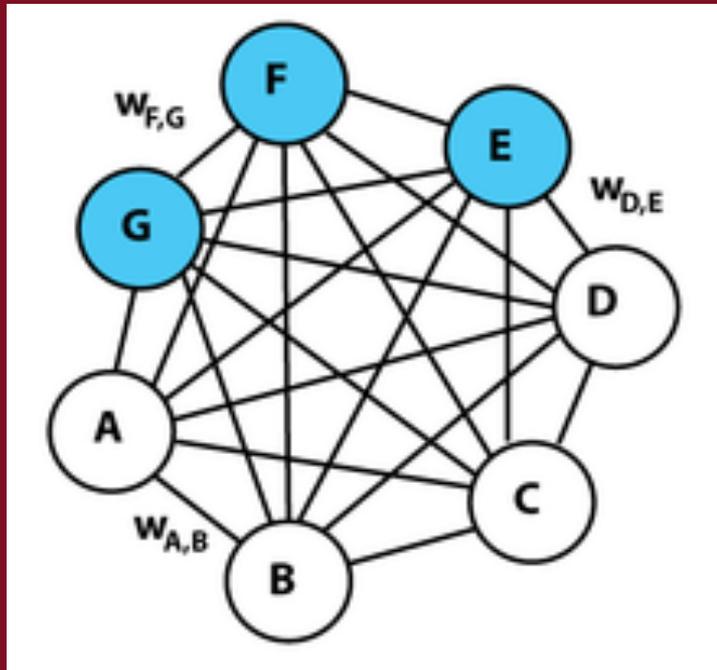
A series of horizontal stripes in various colors (yellow, green, blue, dark blue, orange, red, white, green) running across the bottom of the slide.

Summary of Problem

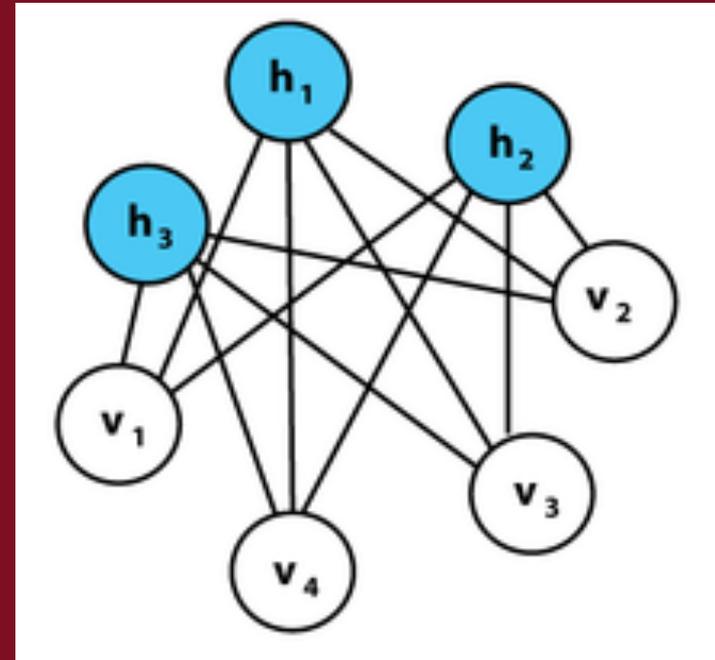
- Pattern Recognition



Background on Deep Boltzmann Machines



Boltzmann Machine



Restricted Boltzmann Machine

Background on Deep Boltzmann Machines

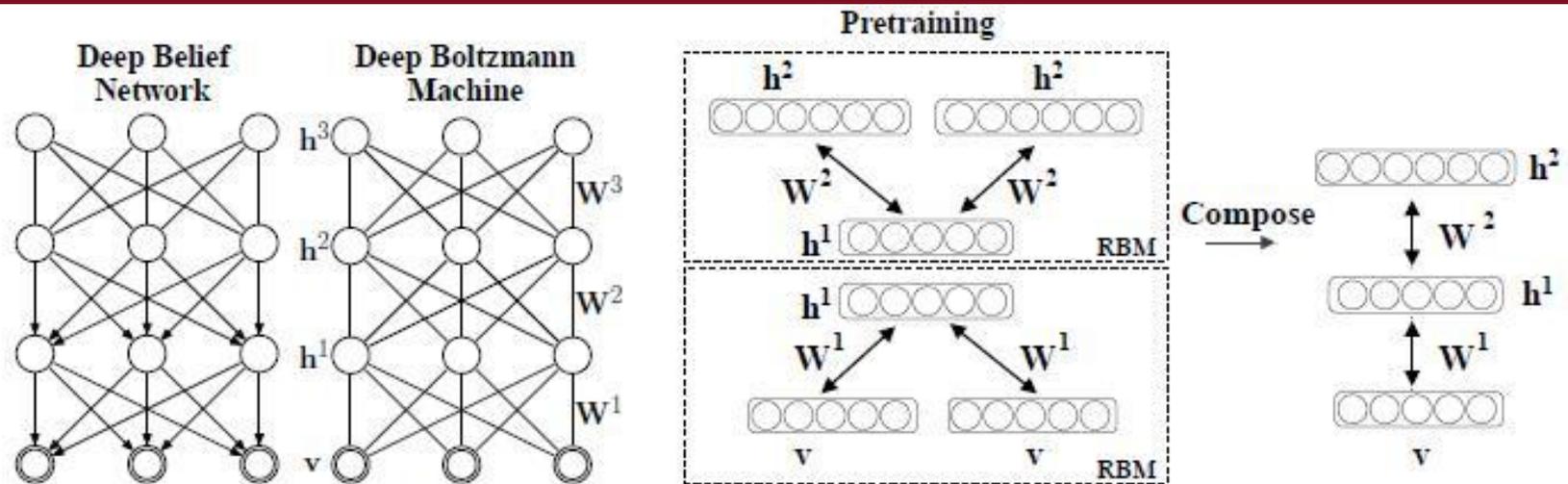


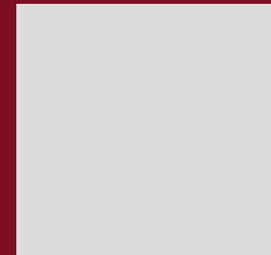
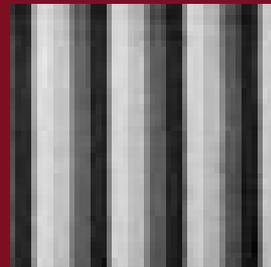
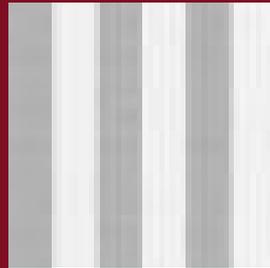
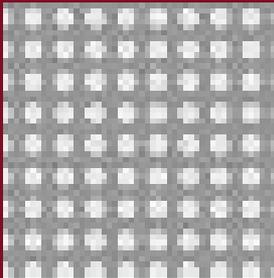
Figure 2: Left: A three-layer Deep Belief Network and a three-layer Deep Boltzmann Machine. Right: Pretraining consists of learning a stack of modified RBM's, that are then composed to create a deep Boltzmann machine.

Our Approach

- Training and testing sets
- Image processing
- Training with RBMs to create images
- Building a DBM to find classification errors

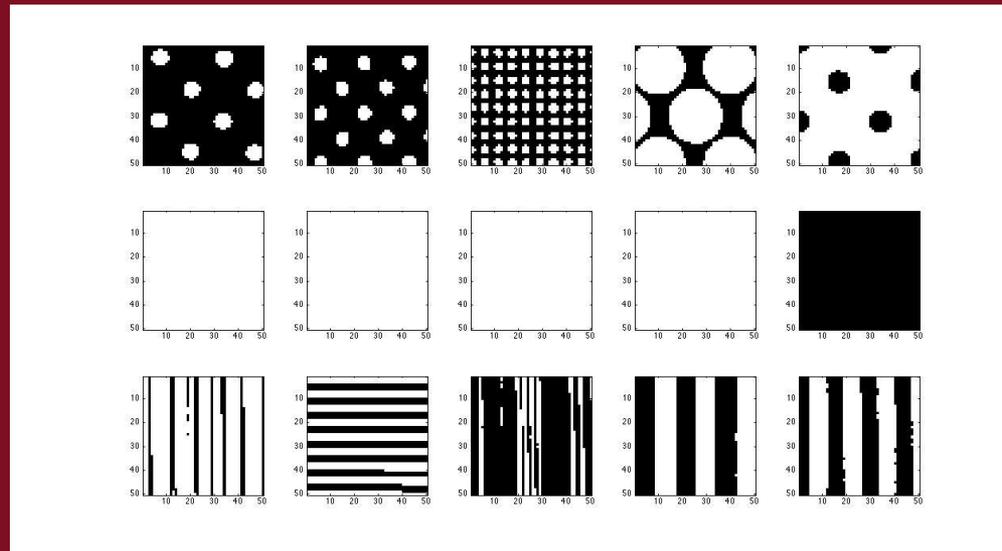
Our Approach-Image Processing

- Manual conversions for uniformity
 - 50x50 pixels
 - Grayscale



Our Approach-Image Processing

- Matlab processing
 - Ultimately convert to binary in .mat file
 - Save classification labels for each image



Our Approach-Neural Network

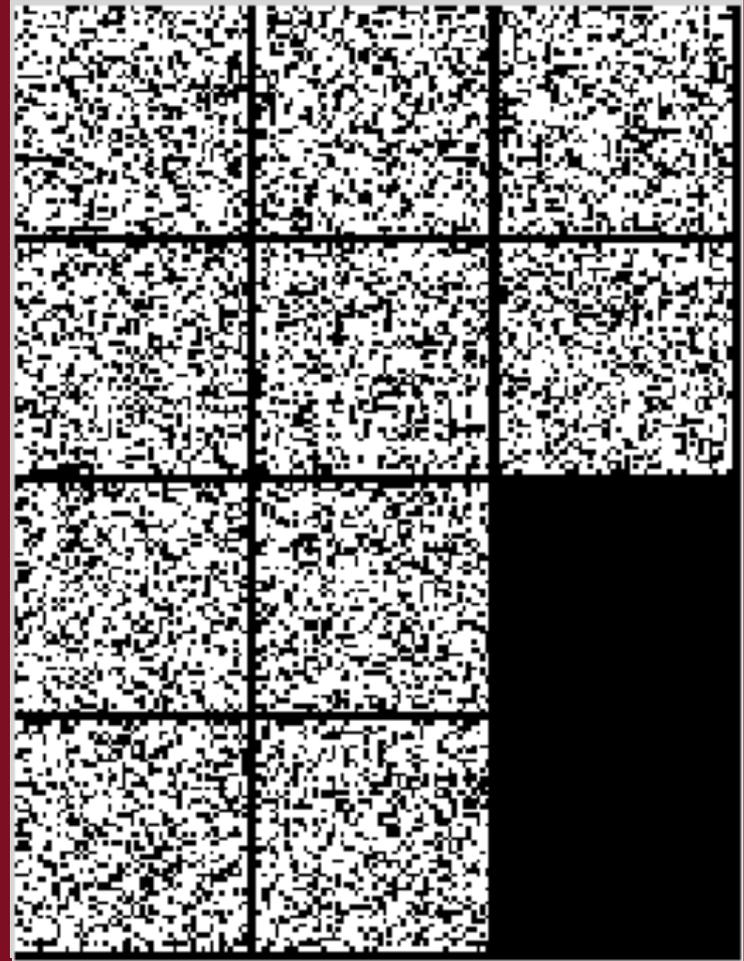
- Pre-train 3 layers of RBMs
 - 500 neurons in the first two layers, 2000 neurons in the last layer

Our Approach-Neural Network

- Making a Deep Boltzmann Machine
 - Use the defining values from the RBMs to classify the training and testing data

Our Results - Training 3 layers of RBMs

- Not the worst...
 - Layer 1:
 - Layer 2:
 - ...and Layer 3:



Our Results - Testing

- Not so good
- 63% misclassification rate (for 3 layers)

Future Work

- Improve code compatibility

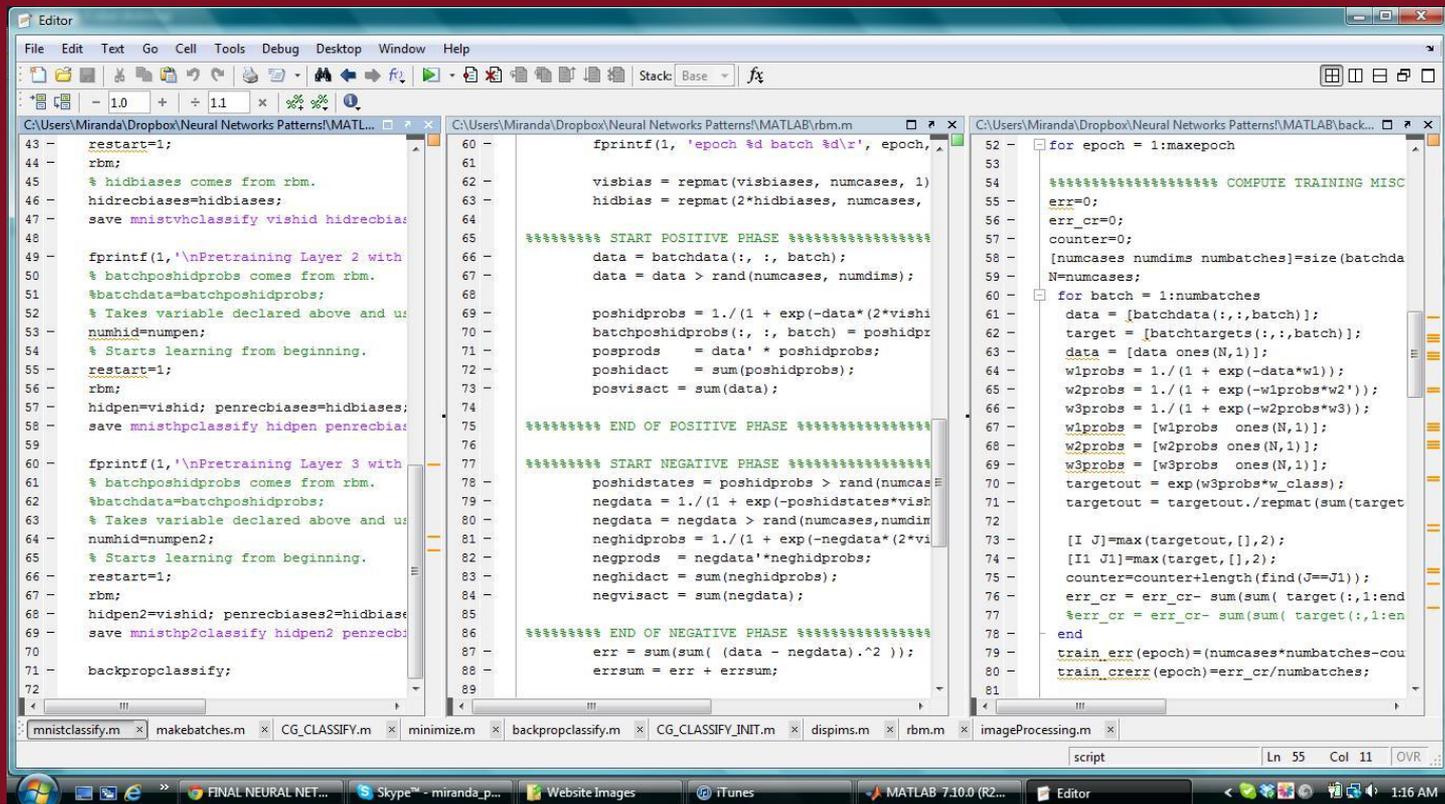
Elegance & Simplicity

Scenario

Quinn was asked to write a function to check an array of floating point numbers to make sure none are zero. The function will be used to check a column vector of coefficients in a scientific computation, which would fail if any of the coefficients were zero.

Future Work

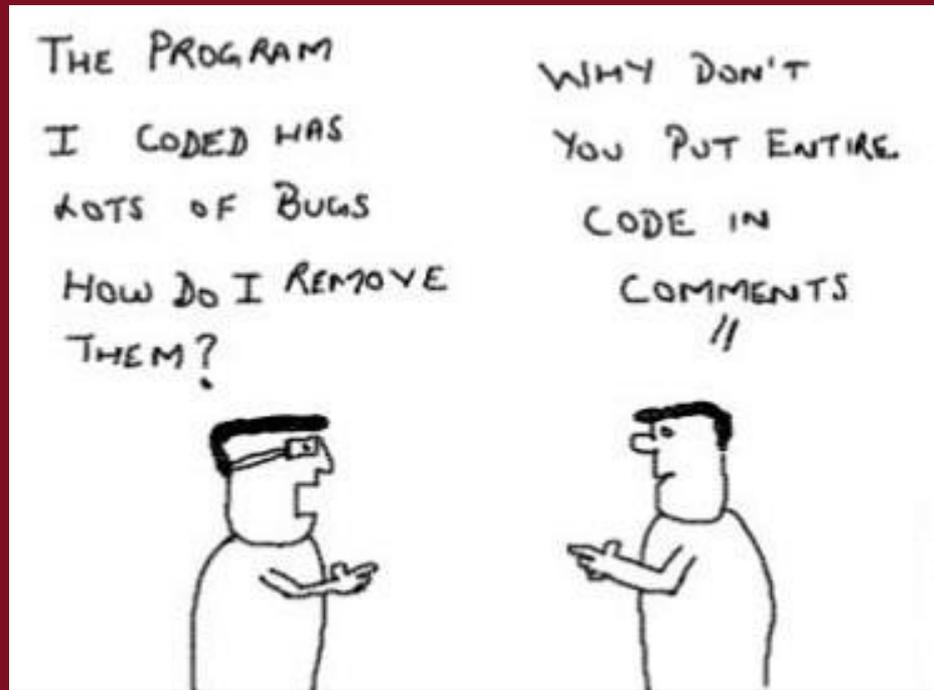
- Comment the Code!



```
43 - restart=1;
44 - rbm;
45 - % hidbiases comes from rbm.
46 - hidrecbiases=hidbiases;
47 - save mnistvhclassify vishid hidrecbia;
48 -
49 - fprintf(1, '\nPretraining Layer 2 with
50 - % batchposhidprobs comes from rbm.
51 - %batchdata=batchposhidprobs;
52 - % Takes variable declared above and u
53 - numhid=numpen;
54 - % Starts learning from beginning.
55 - restart=1;
56 - rbm;
57 - hidpen=vishid; penrecbiases=hidbiases;
58 - save mnisthpclassify hidpen penrecbia;
59 -
60 - fprintf(1, '\nPretraining Layer 3 with
61 - % batchposhidprobs comes from rbm.
62 - %batchdata=batchposhidprobs;
63 - % Takes variable declared above and u
64 - numhid=numpen2;
65 - % Starts learning from beginning.
66 - restart=1;
67 - rbm;
68 - hidpen2=vishid; penrecbiases2=hidbiases;
69 - save mnisthpc2classify hidpen2 penrecbia;
70 -
71 - backpropclassify;
72 -
60 - fprintf(1, 'epoch %d batch %d\r', epoch,
61 -
62 - visbias = repmat(visbiases, numcases, 1)
63 - hidbias = repmat(2*hidbiases, numcases,
64 -
65 -
66 - %%%%%%%%% START POSITIVE PHASE %%%%%%%%%
67 - data = batchdata(:, :, batch);
68 - data = data > rand(numcases, numdims);
69 -
70 - poshidprobs = 1./(1 + exp(-data*(2*vishid
71 - batchposhidprobs(:, :, batch) = poshidpr
72 - posprobs = data' * poshidprobs;
73 - poshidact = sum(poshidprobs);
74 - posvisact = sum(data);
75 -
76 - %%%%%%%%% END OF POSITIVE PHASE %%%%%%%%%
77 -
78 - %%%%%%%%% START NEGATIVE PHASE %%%%%%%%%
79 - poshidstates = poshidprobs > rand(numcas
80 - negdata = 1./(1 + exp(-poshidstates*vish
81 - negdata = negdata > rand(numcases, numdim
82 - neghidprobs = 1./(1 + exp(-negdata*(2*vi
83 - negprobs = negdata' * neghidprobs;
84 - neghidact = sum(neghidprobs);
85 - negvisact = sum(negdata);
86 -
87 - %%%%%%%%% END OF NEGATIVE PHASE %%%%%%%%%
88 - err = sum(sum( (data - negdata).^2 ));
89 - errsum = err + errsum;
52 - for epoch = 1:maxepoch
53 -
54 - %%%%%%%%% COMPUTE TRAINING MISC
55 - err=0;
56 - err_cr=0;
57 - counter=0;
58 - [numcases numdims numbatches]=size(batchda
59 - N=numcases;
60 - for batch = 1:numbatches
61 - data = [batchdata(:, :, batch)];
62 - target = [batchtargets(:, :, batch)];
63 - data = [data ones(N,1)];
64 - w1probs = 1./(1 + exp(-data*w1));
65 - w2probs = 1./(1 + exp(-w1probs*w2'));
66 - w3probs = 1./(1 + exp(-w2probs*w3));
67 - w1probs = [w1probs ones(N,1)];
68 - w2probs = [w2probs ones(N,1)];
69 - w3probs = [w3probs ones(N,1)];
70 - targetout = exp(w3probs*w_class);
71 - targetout = targetout ./ repmat(sum(target
72 -
73 - [I J]=max(targetout, [], 2);
74 - [I1 J1]=max(target, [], 2);
75 - counter=counter+length(find(J==J1));
76 - err_cr = err_cr - sum(sum( target(:, 1:en
77 - %err_cr = err_cr - sum(sum( target(:, 1:en
78 - end
79 - train_err(epoch) = (numcases*numbatches - cou
80 - train_errcr(epoch) = err_cr / numbatches;
81 -
```

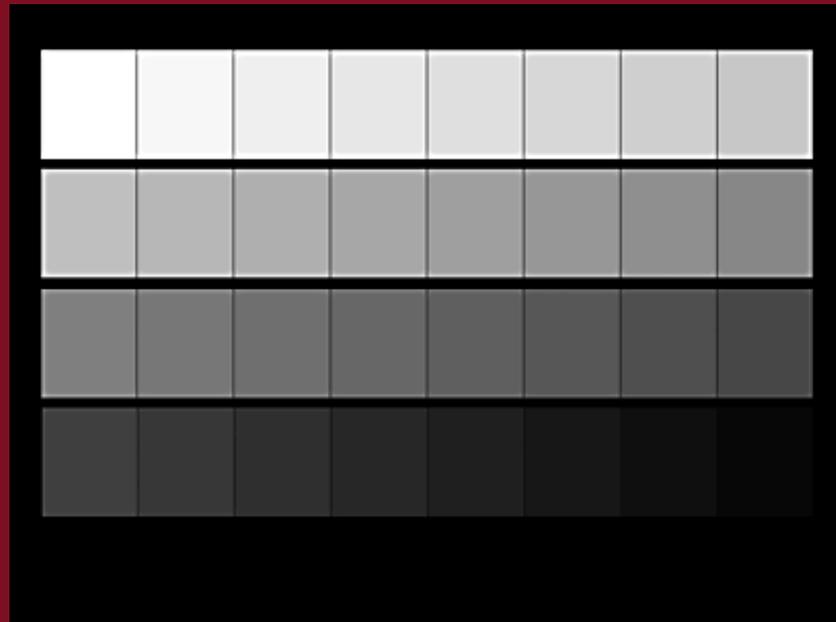
Future Work

- Improve performance
- Use more patterns!



Future Work

- Sanity check - no interference = good results?



References

http://deeplearning.net/software_links/

<http://www.cs.utoronto.ca/~rsalakhu/papers/dbm.pdf>

<http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html>