CS420 Project I

Examination of Pattern Stability in Hopfield Networks

Alexander Saites
3/5/2012
Introduction
In this project, I examined Hopfield networks by imprinting 1 to 50 patterns on a 100-neuron Hopfield network. After each imprinting, I tested each pattern for stability and kept track of the total number of stable patterns and the percentage of unstable patterns. This process was repeated 50 times and the results were averaged.

In a second set of experiments, the program was modified to allow a range of neurons. I used the same experimentation procedure as above for this range of neurons and kept track of the results. Finally, the average number of unstable patterns and the total number of stable patterns were plotted as a surface against the number of patterns imprinted and the number of neurons used. In these experiments, the total number of experiments used before averaging was reduced to 20 in order to increase computation time.

Graphs and Analysis
The following two graphs show the results for the first sets of experiment set. These results show that for 100 Neurons, the number of stable patterns increases linearly until about 15 patterns are stored before dropping back down in a symmetric manner toward zero stability.

![Graph showing stability increases to a fixed number before dropping back to zero.](image)

Figure 1: Stability Increases to a fixed number before dropping back to zero.
The graph of the percentage of unstable patterns of course shows a similar pattern, in that at first, the patterns are stable, but after around 10 patterns are stored, instability rapidly increases until all stored patterns are unstable.

![Graph showing percentage of unstable imprints vs total imprints for 100 neurons, 50 experiments. The graph starts off stable and then quickly becomes unstable as more patterns are added.](image)

**Figure 2: Pattern start off stable, but as more are added, the network quickly becomes unstable**

The following two graphs show the same procedure performed for 200 neurons and show a similar pattern. However, since more neurons are available, more stable patterns are able to be stored.
Figure 3: Stability still rapidly increases and decreases, but more patterns are able to be stored.

Figure 4: The percentage of unstable patterns remains zero for a longer time.
The following graphs show the effects of varying the number of neurons in the network. Unsurprisingly, more neurons mean more stability in stored patterns. Note that the total number of patterns stored (1 to 50) does not change as the number of neurons increases. This allows the total number of stable patterns to eventually reach 50, although the experimentation was not continued up to this point.

The results show the unsurprising general trend that more neurons allow more patterns to be stored. What is slightly more interesting is that this pattern does appear to be linear, implying that in order to store more stable patterns, one would need a linearly increasing number of neurons.

Figure 5: The average number of stable patterns increases linearly with the number of neurons.
Figure 6: The percentage of unstable patterns drops off as more neurons are added.

Code
The program used to experiment was written in Matlab. It consisted of one function to create and test a Hopfield network and one script to drive the program’s experimentation. These two components are listed and described below.

Driver Script

Explanation
The driver script uses a few different values to allow program control. In the “single” version of the program (i.e., multi = false), simply runs the Hopfield function once before plotting the results and labeling the graphs. In the “multi” version of the program, multiple experiments are performed by running the Hopfield code for each set of neurons provided. The results are then plotted as a surface. The program will go from start to end, stepping by step each time.

Code

```matlab
%% Values
num_experiments = 20; % total number of experiments to average over
multi = true; % whether to use multi-neuron experiment or not
start_num_neurons = 100; % irrelevant if multi is false
end_num_neurons = 200; % total number of neurons used if multi is false
```
step = 1; % step between each set of neurons if using multi
num_patterns = 50; % total number of patterns to store

%% Experiment
if( multi )
    % Multi-neuron version
    tot_neuron_range = end_num_neurons - start_num_neurons;
    avg_stable_prints = zeros( tot_neuron_range, num_patterns );
    avg_unstable_prints = zeros( tot_neuron_range, num_patterns );
    for i=start_num_neurons:step:end_num_neurons
        [avg_stable_prints(i,:), avg_unstable_prints(i,:)] = hopfield( num_patterns, i, num_experiments );
        fprintf( '%d neurons complete
', i);
    end

    figure;
    surf( avg_stable_prints );
    set(gca, 'YLim', [start_num_neurons end_num_neurons])
    t = sprintf('%d to %d Neurons, Step = %d, %d Experiments', ... 
                start_num_neurons, end_num_neurons, num_experiments);
    title(t);
    xlabel('Total Number of Patterns (in each experiment)');
    ylabel('Number of neurons');
    zlabel('Average Number of Stable Patterns');
    figure;
    surf( avg_unstable_prints );
    set(gca, 'YLim', [start_num_neurons end_num_neurons])
    t = sprintf('%d to %d Neurons, Step = %d, %d Experiments', ... 
                start_num_neurons, end_num_neurons, num_experiments);
    title(t);
    xlabel('Total Number of Patterns (in each experiment)');
    ylabel('Number of neurons');
    zlabel('Percentage of Unstable Patterns')
else
    % Single-neuron (simple) Version
    [avg_stable_prints, avg_unstable_prints] = hopfield( num_patterns, end_num_neurons, num_experiments );

    figure;
    plot( avg_stable_prints );
    t = sprintf('Average Number Stable Imprints vs Total Imprints: %d Neurons, %d Experiments', ... 
                end_num_neurons, num_experiments);
    title(t);
    xlabel('Total Number of Patterns (in each experiment)');
    ylabel('Average Number of Stable Patterns');

    figure;
    plot( avg_unstable_prints, 'g' );
    t = sprintf('Percentage of Unstable Imprints vs Total Imprints: %d Neurons, %d Experiments', ... 
                end_num_neurons, num_experiments);
    title(t);
    xlabel('Total Number of Patterns (in each experiment)');
    ylabel('Percentage of Unstable Patterns');
end
Hopfield Function

Explanation
The Hopfield function is the main part of the program and actually performs the computation. It works by creating num_patterns number of random binary patterns, computing the weight vector for each of these patterns which is then added to the total weight vector, and finally testing the stability of each pattern currently added to the network. It then records the results for the number of stable and unstable imprints on the network. This process is repeated for num_experiments and the results are averaged together and returned by the function.

Code
function [avg_stable_prints, avg_unstable_prints] = hopfield( num_patterns, num_neurons, num_experiments )

% init matrices
avg_stable_prints = zeros( 1, num_patterns );
avg_unstable_prints = zeros( 1, num_patterns );
blocks = zeros( num_patterns, num_neurons );

% mask is a matrix of 1s with 0s down the diagonals
mask = (eye( num_neurons )-1)*-1;

% do experiments
for e=1:num_experiments
    % create the random patterns
    for i = 1:num_patterns
        blocks(i,:) = (randi( 2, 1, num_neurons )*-2)+3;
    end

    % init
    stable_prints = zeros( 1, num_patterns );
    unstable_prints = zeros( 1, num_patterns );
    all_weights = zeros( num_neurons );

    % compute weights
    for i = 1:num_patterns
        % create the weight matrix
        this_weight = transpose(blocks(i,:)) * blocks(i,:);
        this_weight = this_weight / num_neurons;

        % put zeros on the diagonals
        this_weight = this_weight .* mask;

        % add this weight matrix to the overall weight matrix
        all_weights = all_weights + this_weight;

    % test the imprinted patterns
    for j=1:i
        % compute the local field
        new = blocks(j,:) * all_weights;
    end
end

% stability
for i = 1:num_patterns
    % test the stability
    this_weight = all_weights * blocks(i,:);
    this_weight = this_weight / num_neurons;
    this_weight = this_weight .* mask;

    % add this weight matrix to the overall weight matrix
    all_weights = all_weights + this_weight;
end

% record results
avg_stable_prints = sum(stable_prints, 1) / num_experiments;
avg_unstable_prints = sum(unstable_prints, 1) / num_experiments;
end
new(new<0) = -1;
new(new=-1) = 1;
if( new == blocks(j,:) )
    stable_prints(i) = stable_prints(i) + 1;
end
end
unstable_prints(i) = 1 - (stable_prints(i)/i); % percentage of unstable prints
end
avg_stable_prints = avg_stable_prints + stable_prints;
avg_unstable_prints = avg_unstable_prints + unstable_prints; % avg percentage of unstable prints
end
avg_stable_prints = avg_stable_prints / num_experiments;
avg_unstable_prints = avg_unstable_prints / num_experiments;