Homework 3: Bidirectional Associative Memory

Andy Reagan

February 19, 2015

1 Discussion

From exploring what BAM looks like from the state space, I've found that BAM, when stable, can only map memories to another node and then back to the start. Also, starting at the other node, you have to get mapped back to memory that was stored on the other side, for it to be stable. Otherwise there is no way for the weight matrix W to encode the desired memories.

I'm still trying to make sense of how the weight matrix W is set, but I've gotten this far: recalling the memory B_1 associated with A_1 goes like

$$B_1 = f(A_1 W) = f(A_1(A_1^T B_1 + \cdot A_n^T B_n)) = f(P_{A_1} B_1 + \cdot P_{A_1 A_n} B_n)$$

where the P_A is the projection matrix that projects B on the basis of A. The projection matrix for orthogonal vectors is 0, and the projection matrix of A onto A is the identity, so this will lead to perfect recall if the memories to be stored are indeed a basis of the vector space for their associations.

2 Visualization

I coded up the network in Javascript and made a force layout of the states of the network, drawing links as you move through the network. I can show you (the class) a demo of it running, it's pretty fun. The javascript code is attached, it also relies on a css and an html file which I didn't include.

Check it out live here: `http://andyreagan.github.io/demos/2015-02AANN/BAM.html`.

Starting at any given state in the network, the green dashed lines follow W^T, and the grey lines follow W. It is sometimes tricky, but if you look closely it is always unambiguous how states are updated using these maps.

Here are a couple screen shots from it:

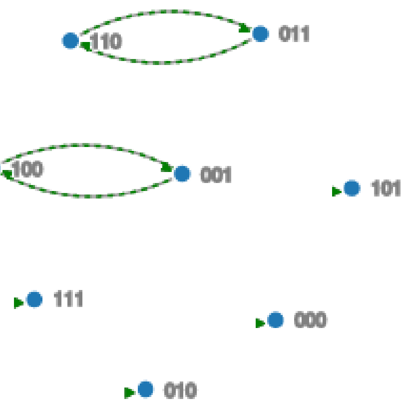

Figure 1: The network for the memories given in class.

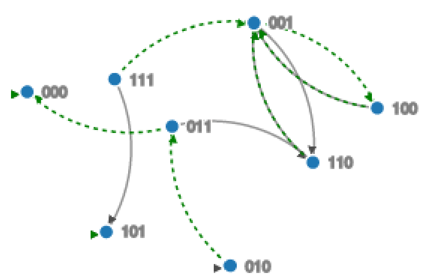

Figure 2: The network for the memories given in class, with a slight change so that the two memories are not symmetric. More interesting!

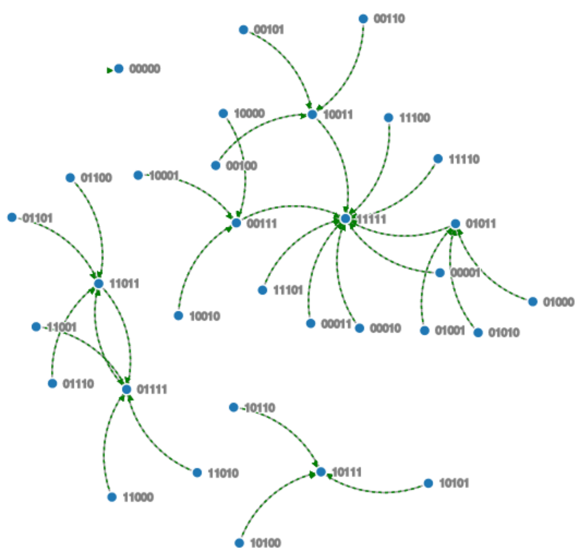

Figure 3: A bigger network having some trouble remembering things. Maybe I coded it wrong...

Full code

```
% bidirectional associative memory (BAM) network
% aka heteroassociative
% accepts an input vector (hard coded here) on set of
% neurons and produces a related but different output
% vector on the other nuerons
%
% stable states are memories that are associated to eachother
% i.e. network can run in either direction
%
% I left all of my work testing the vectorization of creating the
% weight matrix :)
%
% 2015-02-13
% Andy Reagan

% threshold for output activation
mu = 0;

% define activation function
% vectorize!!
perf = @(x) x>(mu.*ones(size(x))) + (x==(mu.*ones(size(x)))).*x;

% training patterns
% are the columns
disp('training␣patterns:')
A = [1,0,0;0,1,0;0,0,1];
B = [0,0,1;0,1,0;1,0,0];
disp(A);
disp(B);

% size of our network
% N = 4;
% N = length(A[:,1]);
N = length(A(:,1));

% set the weights
W = zeros(N);
% with python array access
% W = A[:,1]'*B[:,1] + A[:,2]'*B[:,2] + A[:,3]'*B[:,3];
% with matlab array access
% W = A(:,1)*B(:,1)' + A(:,2)*B(:,2)' + A(:,3)*B(:,3)';
W = (2.*A(:,1)-1)*(2.*B(:,1)-1)' + (2.*A(:,2)-1)*(2.*B(:,2)-1)' + (2.*A(:,3)
    -1)*(2.*B(:,3)-1)';
% disp('weights:');
% disp(W);
% matrix style!!
% W = A'*B;
% also, convert to bipolar
% disp(2.*A-1)
% disp(2.*B-1)
% W = (2.*A-1)*(2.*B-1)';
disp('weights:');
disp(W);
%% test the training patterns

for j=1:length(A(1,:))
    % fetch input
    a = A(:,j)';
    % feed forward
    b = a*W;
    % apply performance rule
    b = perf(b);
    if min(a == perf(b*W')) < 1
        disp('failed␣test')
```

```
        disp('testing␣pattern')
        disp(a);
        disp('this␣is␣b')
        disp(b);
        disp('this␣is␣b␣put␣back␣through')
        disp(perf(b*W'))
    else
        disp('passed␣test')
    end
end

%% let's try to vizualize the network

% hm, going to try doing this in javascript actually!
% so that I can use network viz in d3
```

```
var width = 960,
height = 500;

var color = d3.scale.category20();

var force = d3.layout.force()
    .charge(-120)
    .linkDistance(30)
    .size([width, height]);

var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height);

// var states = 3;

// var A = [[1,0,0],[0,1,0],[0,0,1],];
// var B = [[0,0,1],[0,1,0],[1,1,0],];

// var states = 4;

// var A = [[1,0,0,0],[0,1,0,0],[0,0,1,0],];
// var B = [[0,0,1,0],[0,1,0,0],[1,0,0,0],];

var states = 5;

var A = [[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0],];
var B = [[0,0,1,0,0],[0,1,0,0,0],[1,0,0,0,0],];

var numStates = Math.pow(2,states);

function pad(n, width, z) {
  z = z || '0';
  n = n + '';
  return n.length >= width ? n : new Array(width - n.length + 1).join(z) + n;
}

var nodes = Array(numStates);
for (var i=0; i<numStates; i++) {
    nodes[i] = pad(i.toString(2),states);
}

var nodeslist = nodes.map(function(d) { return {"name": d,}; })

// var W = (2.*A-1)*(2.*B-1)';
var W = math.multiply(math.transpose(math.add(math.multiply(2,A),-1)),math.
     add(math.multiply(2,B),-1));

// little bit harder in JS
function perf2(A) {
    var mu = 0;
    var B = math.zeros(math.size(A));
    for (var i=0; i<math.size(A)[0]; i++) {
        for (var j=0; j<math.size(A)[1]; j++) {
            if (A[i][j] > mu) {
                B[i][j] = 1;
            }
            else {
                B[i][j] = (A[i][j] < mu) ? 0 : A[i][j];
            }
        }
    }
    return B;
}

function perf1(A) {
    var mu = 0;
    var B = math.zeros(math.size(A));
```

```
    for (var i=0; i<math.size(A)[0]; i++) {
        if (A[i] > mu) {
            B[i] = 1;
        }
        else {
            B[i] = (A[i] < mu) ? 0 : A[i];
        }
    }
    return B;
}

function strToArray(a) {
    b = Array(a.length);
    for (var i=0; i<a.length; i++) {
        b[i] = parseInt(a[i]);
    }
    return b;
}

var linkslist = [];

// loop over the nodes, add a link if they map forward to another
for (var i=0; i<numStates; i++) {
    // pull the array out of the string
    a = strToArray(nodes[i]);
    console.log(a);
    b = perf1(math.multiply(a,W));
    console.log(b);
    var j = nodes.indexOf(b.join(''));
    linkslist.push({"source": i,"target": j,"value": 1, "type": "out"});
}

// loop over the nodes, add a link if they map backward to another
for (var i=0; i<numStates; i++) {
    // pull the array out of the string
    a = strToArray(nodes[i]);
    console.log(a);
    b = perf1(math.multiply(a,math.transpose(W)));
    console.log(b);
    var j = nodes.indexOf(b.join(''));
    linkslist.push({"source": i,"target": j,"value": 2, "type": "in",});
}

var graph = {"nodes": nodeslist, "links": linkslist};

force
    .nodes(graph.nodes)
    .links(graph.links)
    .linkDistance(100)
    // .charge(-100)
    .gravity(.05)
    .start();

var path = svg.append("g").selectAll("path")
    .data(force.links())
    .enter()
    .append("path")
    .attr("class", function(d) { return "link " + d.type; })
    .attr("marker-end", function(d) { return "url(#" + d.type + ")"; });
    // .attr("marker-end", function(d) { return "url(#suit)"; });

// var link = svg.selectAll(".link")
//      .data(graph.links)
//      .enter()
//      .append("line")
//      .attr("class", "link")
//      .style("stroke-width", function(d) { return Math.sqrt(d.value); })
//      .style("marker-end",  "url(#suit)"); // Modified line
```

```
var node = svg.selectAll(".node")
    .data(graph.nodes)
    .enter().append("g")
    .attr("class", "node")
    .call(force.drag);

node.append("circle")
    .attr("r", 5)
    .style("fill", function(d) { return color(d.group); })

node.append("text")
    .attr("dx", 10)
    .attr("dy", ".35em")
    .text(function(d) { return d.name; })
    .style("stroke", "grey");

force.on("tick", function() {
    path.attr("d", linkArc);
    // link.attr("x1", function(d) { return d.source.x; })
    //  .attr("y1", function(d) { return d.source.y; })
    //  .attr("x2", function(d) { return d.target.x; })
    //  .attr("y2", function(d) { return d.target.y; });

    d3.selectAll("circle").attr("cx", function(d) { return d.x; })
        .attr("cy", function(d) { return d.y; });

    d3.selectAll("text").attr("x", function(d) { return d.x; })
        .attr("y", function(d) { return d.y; });
});



// Use elliptical arc path segments to doubly-encode directionality.
function tick() {
  path.attr("d", linkArc);
  circle.attr("transform", transform);
  text.attr("transform", transform);
}

function linkArc(d) {
  var dx = d.target.x - d.source.x,
      dy = d.target.y - d.source.y,
      dr = Math.sqrt(dx * dx + dy * dy);
  return "M" + d.source.x + "," + d.source.y + "A" + dr + "," + dr + " 0 0,1 
      " + d.target.x + "," + d.target.y;
}

function transform(d) {
  return "translate(" + d.x + "," + d.y + ")";
}

svg.append("defs").selectAll("marker")
    // .data(["suit", "licensing", "resolved"])
    .data(["out", "in",])
    .enter()
    .append("marker")
    .attr("id", function(d) { return d; })
    .attr("viewBox", "0 -5 10 10")
    .attr("refX", 18)
    .attr("refY", -3)
    .attr("markerWidth", 4)
    .attr("markerHeight", 4)
    .attr("orient", "auto")
    .append("path")
    .attr("d", "M0,-5L10,0L0,5")
    .attr("class",function(d) { return "arrow "+d; });
    // .attr("d", "M0,-5L10,0L0,5 L10,0 L0, -5");
```

```
    // .style("stroke", "#4679BD")
    // .style("opacity", "0.6");
```