Appendix

A. Results of catastrophic forgetting by coupling two multi-layer perceptrons:

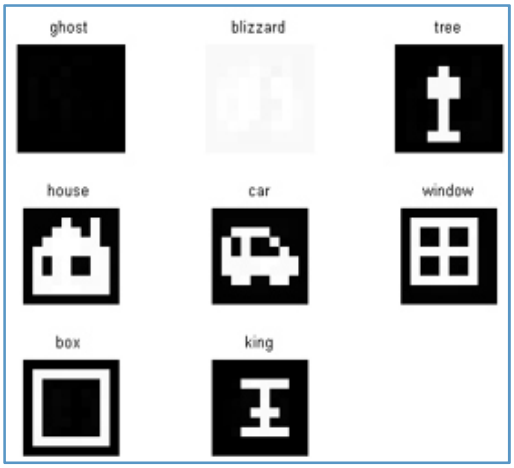

Figure A.1 **–** Original set of images learned by NET1

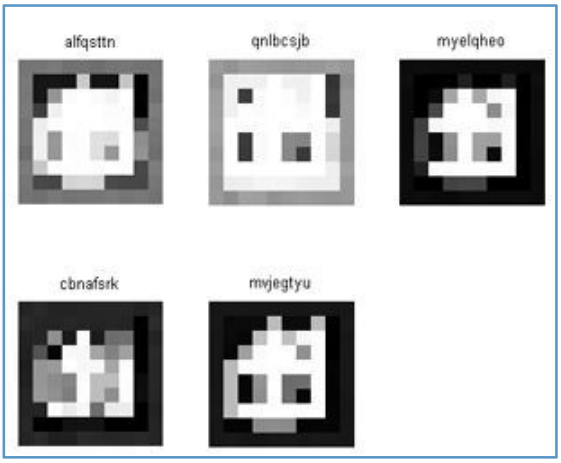

Figure A.2 **–** Pseudo-patterns generated by NET1 after it has been bombarded with noise

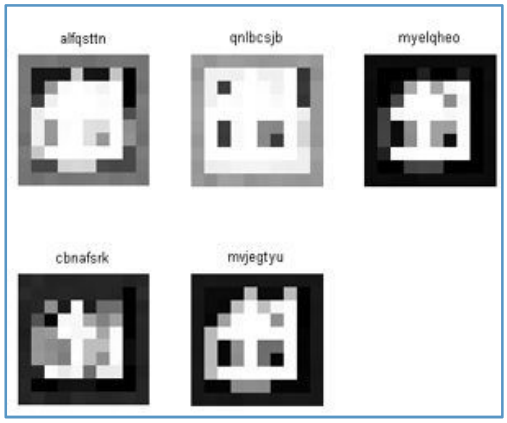

Figure A.3 **–** Pseudo-patterns learned by NET2 and originally generated by NET1

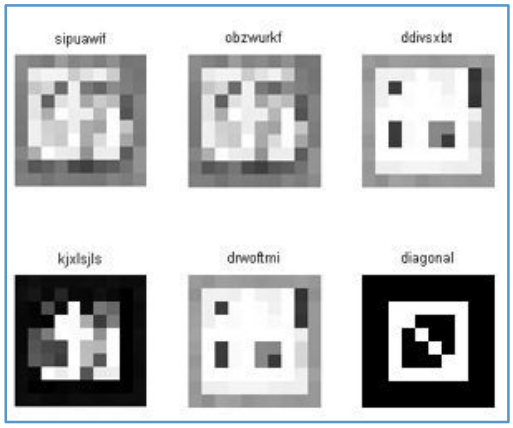

Figure A.4 **–** Pseudo-patterns originally generated by NET2, learned by NET1 along with external input

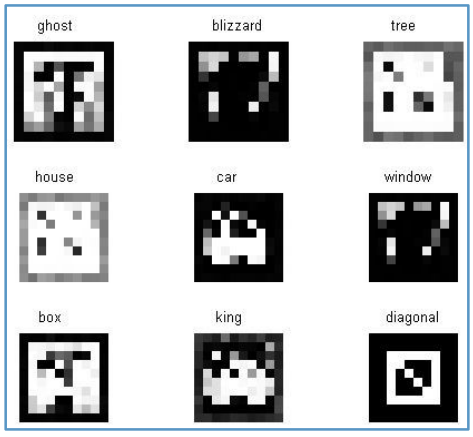

Figure A.5 **–** Output of NET1 after learning pseudo-patterns from NET2 and external input

B. Results of catastrophic forgetting by using:

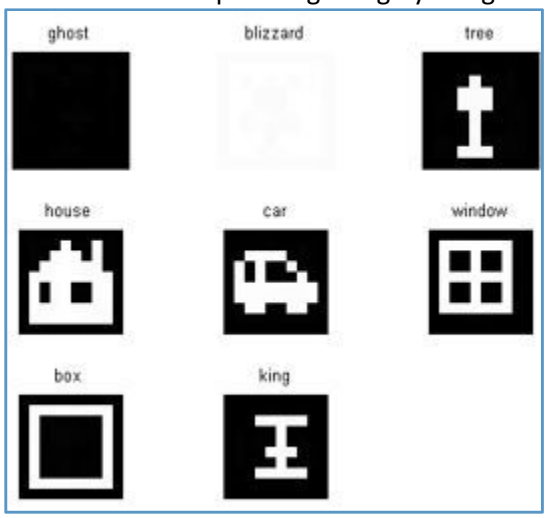

Figure B.1 – Original set of learned images

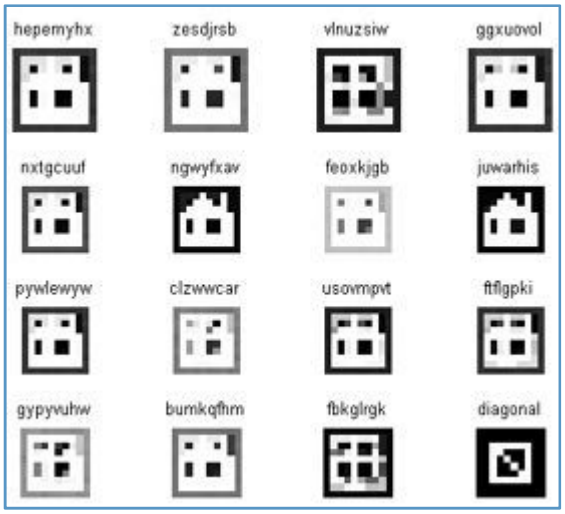

Figure B.2 – Learned pseudo-patterns along with learned image (i.e. diagonal)

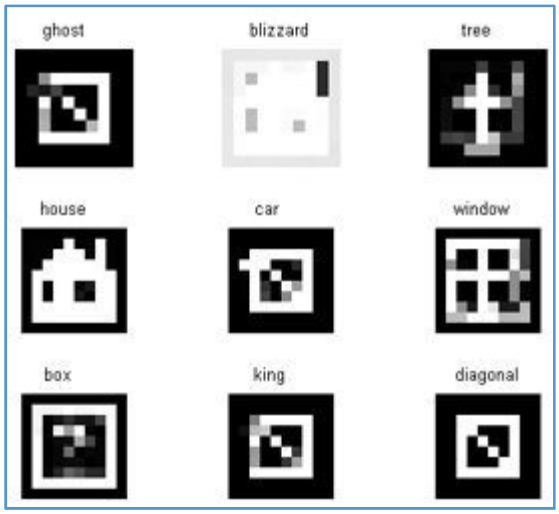

Figure B.3 – Attempt to remember after new image (i.e. diagonal) has been learned

C. Online Learning for Image Learning Module

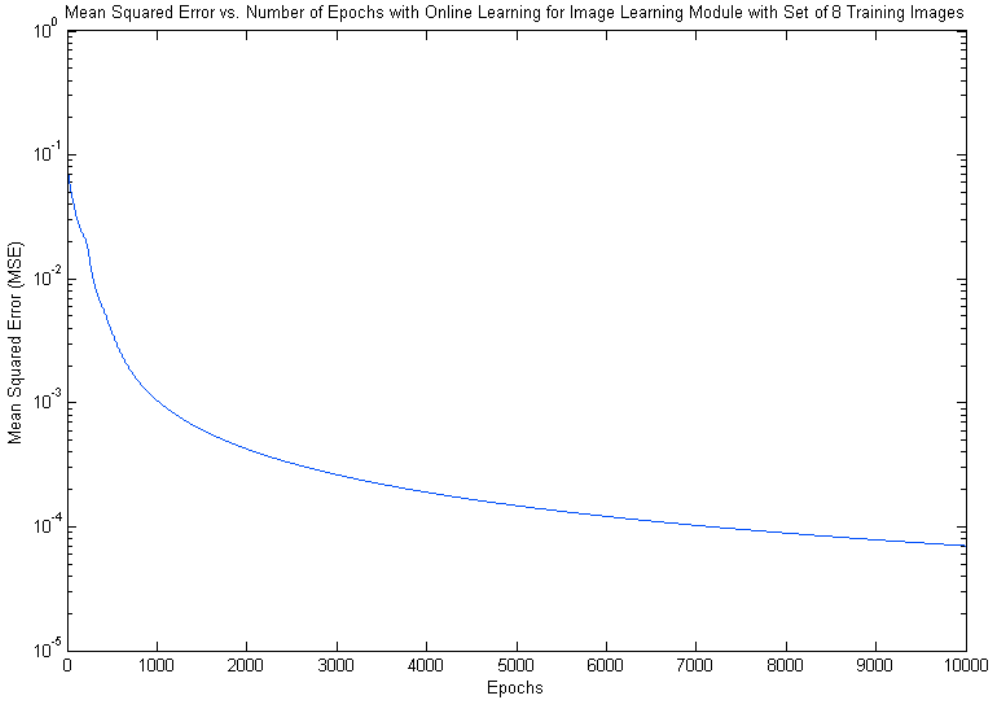

Figure C.1 – Mean squared error vs. number of epochs for image learning module with online learning

D. Batch Learning for Image Learning Module

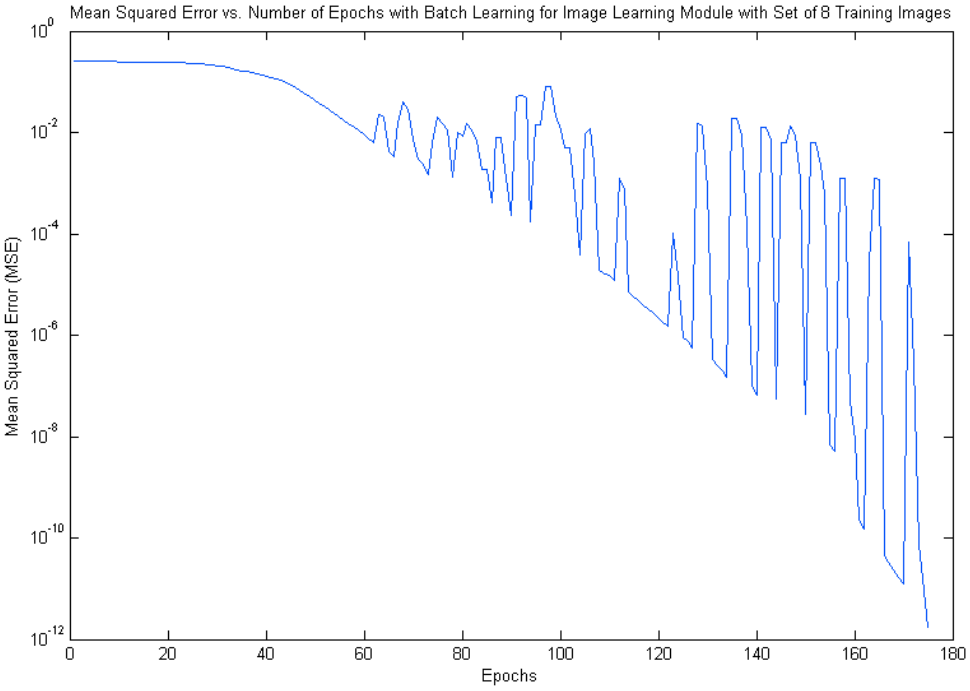

Figure D.1 – Mean squared error vs. number of epochs for image learning module with batch learning

E. Sample Input/Output for Relations Learning Module

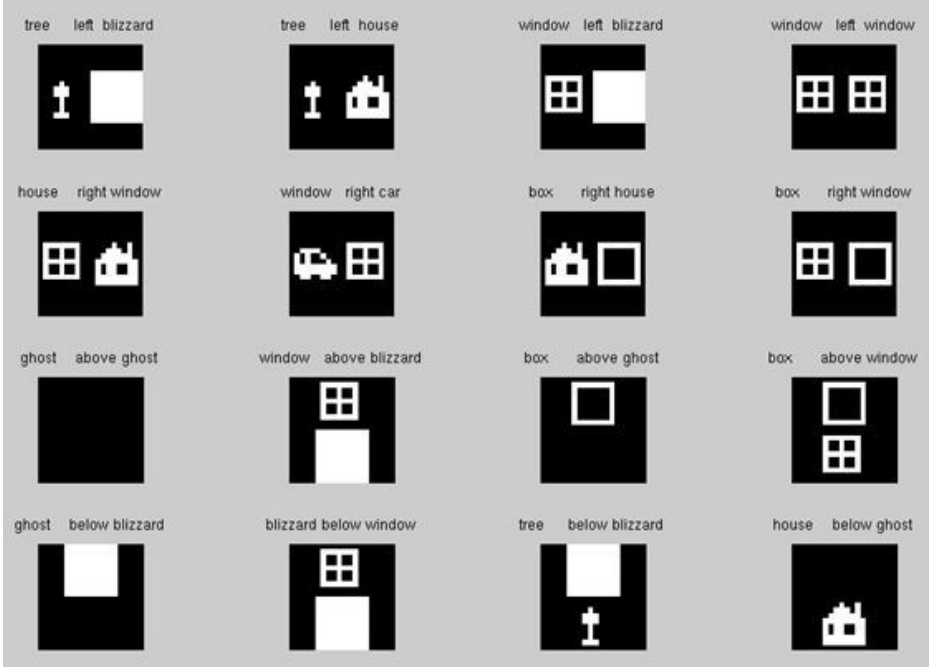

Figure E.1 – Sample input/output associations for relations learning module

F. Test Input/Output for Relations Learning Module

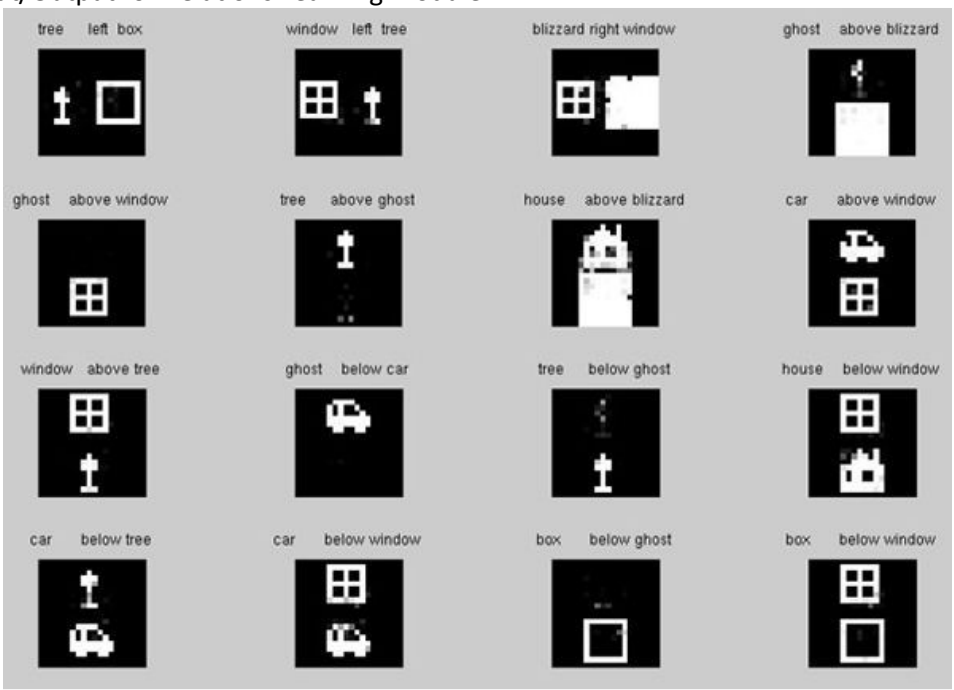

Figure F.1 – Test input/output associations for relations learning module

G. Results of iRprop+ with new weight function

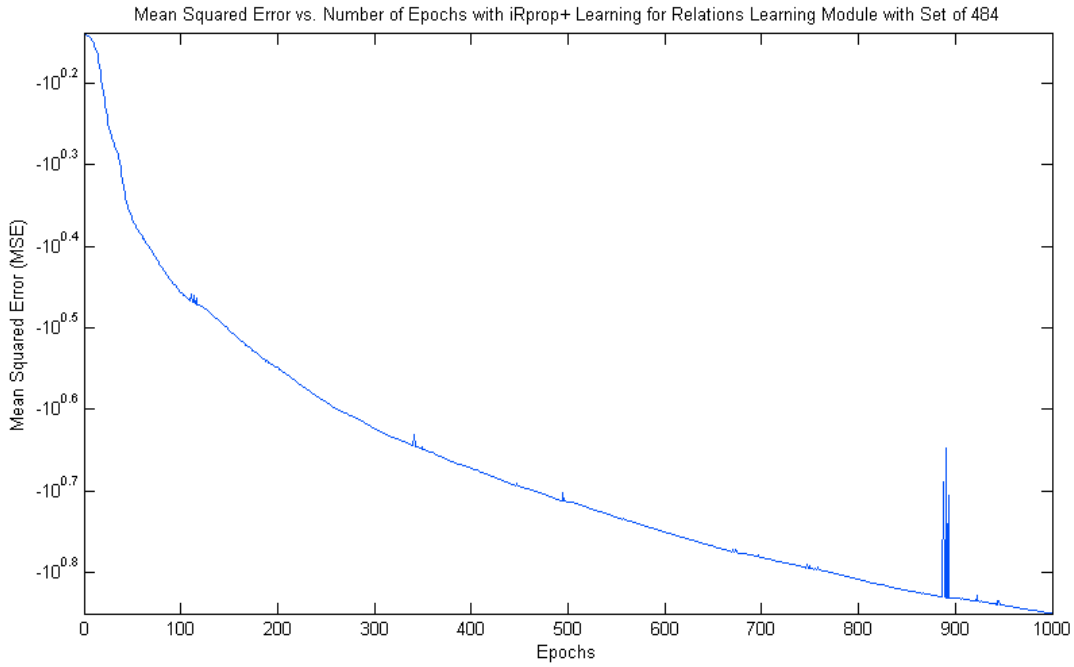

Figure G.1 – Results of iRprop+ with new weight function

H. Implementation Details

We implemented everything in Matlab. Juan used the neural network toolbox to implement the Elman network he used when trying to implement the reverberating network.

Matlab website: http://www.mathworks.com/
Matlab neural network toolbox: http://www.mathworks.com/products/neuralnet/

References

[1] Ans, B., Rousset, S. Avoiding Catastrophic Forgetting by Coupling Two Reverberating Neural Networks. Laboratoire de Psychologie Experimentale (CNRS EP 617), Universite Pierre-Mendes-France, BP 47, 38040 Grenoble cedex 9, France. November 3, 1997. DOI= http://adsabs.harvard.edu/abs/1997CRASG.320..989

[2] Ans, B., Rousset, S., French, R. M., and Musca, S. Self-Refreshing Memory in Artificial Neural Networks: Learning Temporal Sequences without Catastrophic Forgetting. Connectionist Science, Vol. 16, No. 2, June 2004, pp. 71-79. DOI= http://www.citeulike.org/user/chchatham/article/461062

[3] Elman, J. L. Finding Structure in Time. University of California, San Diego. Cognitive Science, 14, pp. 179-211, 1990. DOI= http://homepages.inf.ed.ac.uk/keller/teaching/connectionism/CogSci90-Elman.pdf

[4] Riedmiller, Martin. Rprop - Description and Implementation Details. 1994. DOI= http://citeseer.ist.psu.edu/rd/2171473%2C711503%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs2/20/http:zSzzSzamy.informatik.uos.dezSzriedmillerzSzpublicationszSzrprop.details.pdf/riedmiller94rprop.pdf

[5] Igel, C. Husken, M. Empirical Evaluation of the Improved Rprop Learning Algorithms. Institut fur Neuroir Formatik, Ruhr-Universitdt. Bochum, 44780, Germany. Neurocomputing, 2003, pp. 105-123. DOI= http://www.google.com/url?sa=t&source=web&ct=res&cd=2&url=http%3A%2F%2Fwww.neuroinformatik.ruhr-uni-bochum.de%2FPEOPLE%2Figel%2FEEotIRLA.ps.gz&ei=p3ItSvDqN5S6tgO92NjrCg&usg=AFQjCNFeyyx8F18iD5xCB8jl7WQjnDXeSA&sig2=XmPojoDbXV02WPKY7DO5IQ

```
function [ input, target, wH, wO, errors ] = learnImages(numHiddenNeurons, epochs,
goal_err, lrate, drawrate, color)
% LEARNIMAGES Neural net training for images

if nargin < 1
    numHiddenNeurons = 7;
end

if nargin < 2
    epochs = 1000;
end

if nargin < 3
    goal_err = 10e-12;
end

if nargin < 4
    drawrate = 25;
end

if nargin < 5
    lrate=.8;
end

if nargin < 6
    %color = 'Jet';
    color = 'Gray';
end


% get input
[images, input] = getImages;

% format the input character strings into binary strings
P = str2bin(input);

% image dimensions
[imRows, imCols, numIm] = size(images);

% flatten target images into 1D vectors
T = reshape(images, imRows*imCols, numIm);

% establish starting params
[wH wO] = createWeights(P, T, numHiddenNeurons);
[oO oH] = applyWeights(P, wH, wO);
e = T - oO;
error = mean(mean(e.*e));
e = e .* abs(e);
lasterror = error;

wHf = wH;
wOf = wO;

wHl = wH;
wOl = wO;

% display our current output
window = drawImages(oO,imRows,imCols,numIm,input,color);

etaplus = 1.2;
```

```
etaminus = .5;
maxstep = 1;
minstep = 0;
baselrate = .0000001;       % initial learn rate, should be pretty insensitive
dEwO = zeros(size(wO)); % last step's derivative of the error in the Output layer wrt
output weights
dEwH = zeros(size(wH)); % last step's derivative of the error in the Hidden layer wrt
hidden weights

deltaEwO = ones(size(wO))*baselrate; % the step deltas for the output weights
deltaEwH = ones(size(wH))*baselrate; % the step deltas for the hidden weights

% do training
for  itr =1:epochs
    if error <= goal_err
        break
    end

    % current step's derivates of the errors wrt to weights
    wOb = zeros(size(wO));
    wHb = zeros(size(wH));

    % iterate across all input images
    for i = 1:numIm
        % get derivatives of layer outputs
        dO = dlogsigmoid(oO(:,i));
        dH = dlogsigmoid(oH(:,i));

        % get error slopes for each layer
        eO = dO .* e(:,i) * sum(abs(e(:,i)));
        eH = dH .* (wO' * eO);

        % batch the weight changes
        wOb = wOb - eO * oH(:,i)';
        wHb = wHb - eH * P(:,i)';
    end

    % determine weight changes in output layer
    [maxi maxj] = size(wO);
    signmap = sign(dEwO .* wOb);
    for i = 1:maxi
        for j = 1:maxj
            % determine direction of change
            if signmap(i,j) > 0
                % if sign agrees, push up the learn rate
                deltaEwO(i,j) = min(etaplus*deltaEwO(i,j),maxstep);

                % apply weight change
                wOf(i,j) = wO(i,j) - sign(wOb(i,j))*deltaEwO(i,j);

            elseif signmap(i,j) < 0
                oldWS = deltaEwO(i,j);

                % if signs don't agree, decrease step
                deltaEwO(i,j) = max(etaminus*deltaEwO(i,j),minstep);

                % roll back weight only if overall error increased
                if error > lasterror
                    wOf(i,j) = wO(i,j) + sign(dEwO(i,j))*oldWS;
                end
```

```
                % prevent weight step change next iteration
                wOb(i,j) = 0;

%                 % if error is growing, rollback last step's change
%                 if error > lasterror
%                     wOf(i,j) = wOl(i,j);
%                 end

            else
                % sign is 0

                % apply weight change without change weight step
                wOf(i,j) = wO(i,j) - sign(wOb(i,j))*deltaEwO(i,j);

            end
        end
    end

    % store old weights
    wOl = wO;
    wO = wOf;

%     % adjust weights
%     deltaw = sign(wOb) .* deltaEwO;
%     wO = wO - deltaw;

    dEwO = wOb;




    % determine weight changes in hidden layer
    [maxi maxj] = size(wH);
    signmap = sign(dEwH .* wHb);
    for i = 1:maxi
        for j = 1:maxj
            % determine direction of change
            if signmap(i,j) > 0
                % if sign agrees, push up the learn rate
                deltaEwH(i,j) = min(etaplus*deltaEwH(i,j),maxstep);

                % apply weight change
                wHf(i,j) = wH(i,j) - sign(wHb(i,j))*deltaEwH(i,j);

            elseif signmap(i,j) < 0
                oldWS = deltaEwH(i,j);

                % if signs don't agree, decrease step
                deltaEwH(i,j) = max(etaminus*deltaEwH(i,j),minstep);

                % rollback weight only if error increased
                if error > lasterror
                    wHf(i,j) = wH(i,j) + sign(dEwH(i,j))*oldWS;
                end

                % prevent weight step change next iteration
                wHb(i,j) = 0;

%                 % if error is growing, rollback last step's change
```

```
%                 if error > lasterror
%                     wHf(i,j) = wHl(i,j);
%                 end
            else
                % sign is 0

                % apply weight change without adjusting weightstep
                wHf(i,j) = wH(i,j) - sign(wHb(i,j))*deltaEwH(i,j);
            end
        end
    end

    % store old weights
    wHl = wH;
    wH = wHf;

%     % adjust weights
%     deltaw = sign(wHb) .* deltaEwH;
%     wH = wH - deltaw;

    dEwH = wHb;

%     wO = wO - .0004 * wOb;
%     wH = wH - .0004 * wHb;


    % get new outputs
    [oO oH] = applyWeights(P, wH, wO);

    % calculate new error
    lasterror = error;
    e = T - oO;
    e = e .* abs(e);
    error = mean(mean(e.*e));

    errors(itr) = error;

    disp(sprintf('Iteration :%5d        mse :%12.10f%',itr,error));

    % every once in a while, visualise the outputs
    if mod(itr,drawrate) == 0 && drawrate > 0
        window = drawImages(oO,imRows,imCols,numIm,input,color,window);
    end
end

% display the final output
drawImages(oO,imRows,imCols,numIm,input,color,window);

% assign output
input = P;
target = T;



function [ trainset, testset, wO, wH, imageO, imageH, errors ] = learnRelations( imageH,
imageO, cap )
%LEARNRELATIONS Summary of this function goes here
%   Detailed explanation goes here

% get image module weights, if we weren't given them
```

```
if nargin < 2
    [scratch scratch imageH imageO] = learnImages;
end

% params
numHiddenNeurons = 50;
epochs = 1000;
goal_err = 10e-7;
drawrate = 25;
color = 'Gray';
testsetsize = 16;
imagedisplaysize = 16;

errors = zeros(epochs,1);

% power of our modified error function
pow = 6;

% get inputs
[images phrases] = getRelations(cap);


% format the input character strings into binary strings
P = str2bin(phrases);

% image dimensions
[imRows, imCols, numIm] = size(images);
flatimlen = imRows*imCols;

% flatten target images into 1D vectors
T = reshape(images, flatimlen, numIm);


% restrict input
trainset = randsample(1:numIm,numIm-testsetsize,false);
P = P(:,trainset);
T = T(:,trainset);
phrases = phrases(trainset);

testset = setdiff(1:numIm,trainset);

sim = sort(randsample(1:floor(numIm-testsetsize),imagedisplaysize,false));
[eim nim] = size(sim);


% establish starting params
[phraselen numphrases] = size(P);
[modulehidden modinlen] = size(imageH);
[modoutlen scratch] = size(imageO);

a =  0.05;
b = -0.05;

% generate random weights
%numHiddenNeurons = modoutlen*2 + phraselen - 2 * modinlen;
wH = a + (b-a)*rand(numHiddenNeurons,modoutlen*2 + phraselen - 2 * modinlen);
wO = a + (b-a)*rand(flatimlen,numHiddenNeurons);

wHf = wH;
wOf = wO;
```

```
wHl = wH;
wOl = wO;

% apply the image modules
[oM oMH] = applyWeights(P(1:modinlen,:), imageH, imageO);
[oN oNH] = applyWeights(P((phraselen-modinlen+1):phraselen,:), imageH, imageO);

% consolidate hidden layer input
oMNP = [oM ; P((modinlen+1):(phraselen-modinlen),:) ; oN ];

% get hidden layer output
oH = logsigmoid(wH*oMNP);

% get ouput layer output
oO = logsigmoid(wO*oH);

% calculate error
e = T - oO;
error = mean(mean(e.*e));
e = e.*abs(e).^(pow-2);
lasterror = error;

% display our current output
if drawrate > 0
    window = drawImages(oO(:,sim),imRows,imCols,nim,phrases(sim),color);
end

% initial params
etaplus = 1.2;
etaminus = .5;
maxstep = 50;
minstep = 0;
baselrate = .0001;       % initial learn rate, should be pretty insensitive
dEwO = zeros(size(wO)); % last step's derivative of the error in the Output layer wrt
output weights
dEwH = zeros(size(wH)); % last step's derivative of the error in the Hidden layer wrt
hidden weights

deltaEwO = ones(size(wO))*baselrate; % the step deltas for the output weights
deltaEwH = ones(size(wH))*baselrate; % the step deltas for the hidden weights


wHbest = wH;
wObest = wO;
errorbest = error;


iterset = randsample(1:numphrases,numphrases);

% do training
for  itr =1:epochs
    if error <= goal_err
        break
    end

    % current step's derivates of the errors wrt to weights
    wOb = zeros(size(wO));
    wHb = zeros(size(wH));
```

```
   %iterset = randsample(1:numphrases,floor(numphrases/16));

    % iterate across all input images
    for i = iterset
        % get derivatives of layer outputs
        dO = dlogsigmoid(oO(:,i));
        dH = dlogsigmoid(oH(:,i));

        % get error slopes for each layer
        eO = dO .* e(:,i);
        eH = dH .* (wO' * eO);

        % batch the weight changes
        wOb = wOb - eO * oH(:,i)';
        wHb = wHb - eH * oMNP(:,i)';
    end

    % determine weight changes in output layer
    [maxi maxj] = size(wO);
    signmap = sign(dEwO .* wOb);
    for i = 1:maxi
        for j = 1:maxj
            % determine direction of change
            if signmap(i,j) > 0
                % if sign agrees, push up the learn rate
                deltaEwO(i,j) = min(etaplus*deltaEwO(i,j),maxstep);

                % apply weight change
                wOf(i,j) = wO(i,j) - sign(wOb(i,j))*deltaEwO(i,j);

            elseif signmap(i,j) < 0
                oldWS = deltaEwO(i,j);

                % if signs don't agree, decrease step
                deltaEwO(i,j) = max(etaminus*deltaEwO(i,j),minstep);

                % roll back weight only if overall error increased
                if error > lasterror
                    wOf(i,j) = wO(i,j) + sign(dEwO(i,j))*oldWS;
                end

                % prevent weight step change next iteration
                wOb(i,j) = 0;

            else
                % sign is 0

                % apply weight change without change weight step
                wOf(i,j) = wO(i,j) - sign(wOb(i,j))*deltaEwO(i,j);

            end
        end
    end


    % store old weights
    wOl = wO;
    wO = wOf;

%     % adjust weights
```

```
%     deltaw = sign(wOb) .* deltaEwO;
%     wO = wO - deltaw;

    dEwO = wOb;



    % determine weight changes in hidden layer
    [maxi maxj] = size(wH);
    signmap = sign(dEwH .* wHb);
    for i = 1:maxi
        for j = 1:maxj
            % determine direction of change
            if signmap(i,j) > 0
                % if sign agrees, push up the learn rate
                deltaEwH(i,j) = min(etaplus*deltaEwH(i,j),maxstep);

                % apply weight change
                wHf(i,j) = wH(i,j) - sign(wHb(i,j))*deltaEwH(i,j);

            elseif signmap(i,j) < 0
                oldWS = deltaEwH(i,j);

                % if signs don't agree, decrease step
                deltaEwH(i,j) = max(etaminus*deltaEwH(i,j),minstep);

                % rollback weight only if error increased
                if error > lasterror
                    wHf(i,j) = wH(i,j) + sign(dEwH(i,j))*oldWS;
                end

                % prevent weight step change next iteration
                wHb(i,j) = 0;
            else
                % sign is 0

                % apply weight change without adjusting weightstep
                wHf(i,j) = wH(i,j) - sign(wHb(i,j))*deltaEwH(i,j);
            end
        end
    end

    % store old weights
    wHl = wH;
    wH = wHf;

%     % adjust weights
%     deltaw = sign(wHb) .* deltaEwH;
%     wH = wH - deltaw;

    dEwH = wHb;

%     wO = wO - .0004 * wOb;
%     wH = wH - .0004 * wHb;


    % get new outputs
    % apply the image modules
    [oM oMH] = applyWeights(P(1:modinlen,:), imageH, imageO);
```

```
    [oN oNH] = applyWeights(P((phraselen-modinlen+1):phraselen,:), imageH, imageO);

    % consolidate hidden layer input
    oMNP = [oM ; P((modinlen+1):(phraselen-modinlen),:) ; oN ];

    % get hidden layer output
    oH = logsigmoid(wH*oMNP);

    % get ouput layer output
    oO = logsigmoid(wO*oH);


    % calculate new error
    lasterror = error;
    e = T - oO;
    error = mean(mean(e.*e));
    e = e.*abs(e).^(pow-2);

    errors(itr) = error;

    if error < errorbest
        wHbest = wH;
        wObest = wO;
        besterror = error;
    end

    disp(sprintf('Iteration :%5d        mse :%12.10f%',itr,error));

    % every once in a while, visualise the outputs
    if mod(itr,drawrate) == 0 && drawrate > 0
        window = drawImages(oO(:,sim),imRows,imCols,nim,phrases(sim),color,window);
    end
end

% display the final output
if drawrate > 0
    window = drawImages(oO(:,sim),imRows,imCols,nim,phrases(sim),color,window);
end

wH = wHbest;
wO = wObest;



function [ images, input ] = addImageToLearn(strInput, imgTarget)

% configure
numHiddenNeurons = 7;
epochs = 10000;
goal_err = 10e-12;
drawrate = 100;
lrate=.2;
color = 'Gray';

numPseudoPatterns = 5;

% get original weights
if exist('imageH')
else
    [oInput oTarget imageH imageO] = learnImages(numHiddenNeurons, epochs, goal_err,
lrate, drawrate, color);
```

```
end

[len num] = size(oInput);
[tLen tNum] = size(oTarget);

% generate input for NET 1
pseudoInput = randsample('abcdefghijklmnopqrstuvwxyz', (len/7)*numPseudoPatterns, true,
[]);
pseudoInput = reshape(str2bin(pseudoInput), len, numPseudoPatterns);

% get output for pseudo-patterns
[pO pH] = applyWeights(pseudoInput, imageH, imageO);

% map strings to images
[len numin] = size(pseudoInput);
[imRows imCols scratch] = size(imgTarget);
for i = 1:numin
    strIn(i) = { bin2str(reshape(pseudoInput(:,i), 7, len/7)) };
    imgIn(:,:,i) = reshape(pO(:,i), imRows, imCols);
end

% display pseudo-patterns
[oO oH] = applyWeights(pseudoInput, imageH, imageO);
window = drawImages(oO,imRows,imCols,numin,strIn,color);

% generate input for NET 2
% pseudoInput2 = randsample('abcdefghijklmnopqrstuvwxyz', (len/7)*numPseudoPatterns, true,
 []);
% pseudoInput2 = reshape(str2bin(pseudoInput2), len, numPseudoPatterns);
%
% % map strings to images
% [len numin] = size(pseudoInput2);
% [imRows imCols scratch] = size(imgTarget);
% for i = 1:numin
%     strIn(i) = { bin2str(reshape(pseudoInput2(:,i), 7, len/7)) };
%     imgIn(:,:,i) = reshape(oO(:,i), imRows, imCols);
% end

% have NET 2 learn based on new input and pseudo-pattern output from NET 1
[wH wO] = createWeights(pseudoInput, oO, numHiddenNeurons);
window = figure;
% [oO, wH, wO] = doOnlineTraining(numHiddenNeurons, epochs, goal_err, lrate, drawrate,
color, strIn, imgIn, pseudoInput2, oO, wH, wO, window);
[oO, wH, wO] = doBatchTraining(numHiddenNeurons, epochs, goal_err, lrate, drawrate, color,
 strIn, imgIn, pseudoInput, oO, wH, wO, window);


% feed NET 1 with external input and pseudo-pattern output from NET 2
pseudoInput2 = randsample('abcdefghijklmnopqrstuvwxyz', (len/7)*numPseudoPatterns, true,
[]);
pseudoInput2 = reshape(str2bin(pseudoInput2), len, numPseudoPatterns);
[oO oH] = applyWeights(pseudoInput2, wH, wO);

% test dimensions to make sure they match
P = str2bin(strInput);
[iStrLen iStrNum] = size(P);
if iStrLen ~= len
    error('Input string lengths do not match!');
end
```

```
[iImgRows, iImgCols, iNumImg] = size(imgTarget);
T = reshape(imgTarget, iImgRows*iImgCols, iNumImg);
[iImgLen, iImgNum] = size(T);
if iImgLen ~= tLen
    error('Input image dimensions do not match!');
end

% join pseudo-patterns and new input
pI = [pseudoInput2 P];
pO = [oO T];

% map strings to images
[len numin] = size(pI);
[imRows imCols scratch] = size(imgTarget);
for i = 1:numin
    strIn(i) = { bin2str(reshape(pI(:,i), 7, len/7)) };
    imgIn(:,:,i) = reshape(pO(:,i), imRows, imCols);
end

% display pseudo-patterns & input
[oO oH] = applyWeights(pI, imageH, imageO);
window = drawImages(oO,imRows,imCols,numin,strIn,color);

% train on pseudo-patterns & input
[oO, wH, wO] = doBatchTraining(numHiddenNeurons, epochs, goal_err, lrate, drawrate, color,
 strIn, imgIn, pI, pO, imageH, imageO, window);
% [oO, wH, wO] = doOnlineTraining(numHiddenNeurons, epochs, goal_err, lrate, drawrate,
color, strIn, imgIn, pI, pO, imageH, imageO, window);


% display learned input & pseudo-patterns
drawImages(oO,imRows,imCols,numin,strIn,color,window);


% use new weights to display original images - hopefully without catastrophic forgetting!
pI = [oInput P];    % pO = [oTarget T];

% get strings
[len numin] = size(pI);
for i = 1:numin
    strIn(i) = { bin2str(reshape(pI(:,i), 7, len/7)) };
end

[oO oH] = applyWeights(pI, wH, wO);
drawImages(oO,imRows,imCols,numin,strIn,color);
```