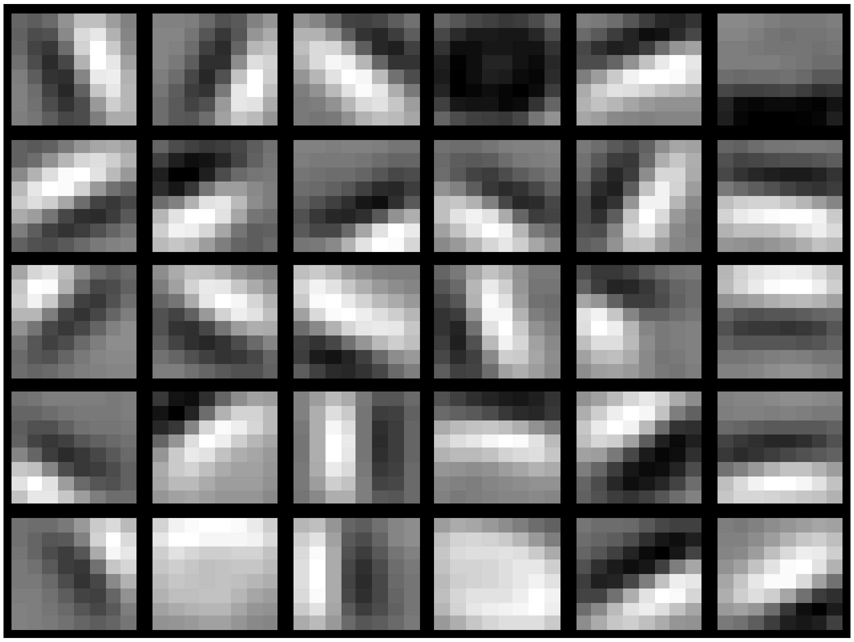

```
function h_x = ncode(data, alph)
  % learning rates
  bt = 5;

  % other algorithm parameters
  rho = -0.996;
  lambda = 0.002;

  % general variables
  img_size = 512;
  sample_size = 8; % 8x8 patch of image
  iterations = 1e8;
  status_output_interval = 1e3;
  file_output_interval = iterations / 100;
  out_path = 'bases/';
  out_ext = 'dat';

  % initializing weights matrix and bias values
  W_1 = rand(30, 64) / sqrt(64);
  W_2 = rand(64, 30) / sqrt(30);

  b_1 = zeros(30, 1);
  b_2 = zeros(64, 1);

  % initializing running rho estimate vector
  rho_est = zeros(30, 1);

  for i = 1:iterations
    %-- feedforward pass (computing activations) --
    x = training_example(sample_size, data, img_size);

    % hidden layer
    z_2 = W_1 * x + b_1;
    a_2 = tanh(z_2);

    % output layer
    z_3 = W_2 * a_2 + b_2;
    a_3 = tanh(z_3);

    % -- computing errors and node responsibilities --
    d_3 = -(x - a_3) .* (1 - a_3.^2);
    d_2 = (W_2' * d_3) .* (1 - a_2.^2);

    % -- updating parameters (gradient descent) --
    W_1 = W_1 - alph * (d_2 * x' + lambda * W_1);
    b_1 = b_1 - alph * d_2;

    W_2 = W_2 - alph * (d_3 * a_2' + lambda * W_2);
    b_2 = b_2 - alph * d_3;

    % updating running rho estimate vector
    rho_est = 0.999 * rho_est + 0.001 * a_2;

    % updating hidden layer intercept terms based on rho estimate vector
    b_1 = b_1 - alph * bt * (rho_est - rho);

    if rem(i, status_output_interval) == 0
      disp(i / iterations);
    end

    if rem(i, file_output_interval) == 0
      write_weights_matrix(W_1, alph, out_ext);
    end
  end

  write_weights_matrix(W_1, alph, out_ext);
end



function [] = write_weights_matrix(W, alph, out_ext)
  file = ['bases' filesep ...
          'weights1-a(' num2str(alph) ')-' ...
          datestr(now, 'HHMMSS') '.' out_ext];
  dlmwrite(file, W);
  disp('w');
end


function x = training_example(sample_size, data, img_size)
  img = random_image(data, img_size);

  patch_i = ceil(rand(2, 1) * (img_size - sample_size + 1));

  % obtaining a random sample patch
  r_i = patch_i(1);
  r_f = r_i + sample_size - 1;
  c_i = patch_i(2);
  c_f = c_i + sample_size - 1;

  r_patch = img(r_i:r_f, c_i:c_f);

  % vectorizing r_patch
  x = r_patch(:);
end



function m = random_image(data, img_size)
  random_index = ceil(rand(1) * 10);
  m = image_matrix(random_index, data, img_size);
end


% returns matrix m containing raw pixel values for image
% assumes img_size is the row count of a single image
% assumes data contains the concatenation of all images
function m = image_matrix(i, data, img_size)
  start_row = (i - 1) * img_size + 1;
  end_row = start_row + img_size - 1;
  m = data(start_row:end_row,:);
end
```

ncode.m