Contents

CS294A/CS294W Programming Assignment Starter Code

```
%  Instructions
%  ------------
%
%  This file contains code that helps you get started on the
%  programming assignment. You will need to complete the code in sampleIMAGES.m,
%  sparseAutoencoderCost.m and computeNumericalGradient.m.
%  For the purpose of completing the assignment, you do not need to
%  change the code in this file.
%
%%======================================================================
```

STEP 0: Here we provide the relevant parameters values that will

allow your sparse autoencoder to get good filters; you do not need to
change the parameters below.

```
% visibleSize = 8*8;   % number of input units
% hiddenSize = 25;     % number of hidden units
% sparsityParam = 0.01;   % desired average activation of the hidden units.
%                      % (This was denoted by the Greek alphabet rho, which looks like a lower-case "p",
%                     %  in the lecture notes).
% lambda = 0.0001;     % weight decay parameter
% beta = 3;            % weight of sparsity penalty term

clear;clc;
close all;

% parameters for Vectorization:Step 2
visibleSize = 28*28;
hiddenSize = 196;
sparsityParam = 0.1;
lambda = 3e-3;
numPatches = 10000;
beta = 3;

%%======================================================================
```

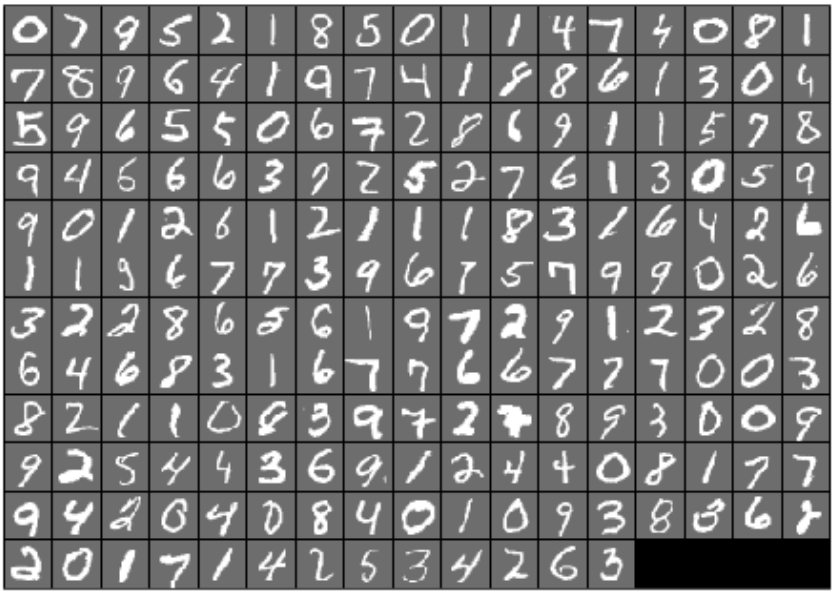
STEP 1: Implement sampleIMAGES

After implementing sampleIMAGES, the display_network command should
display a random sample of 200 patches from the dataset

```
images = loadMNISTImages('train-images-idx3-ubyte');
patches=images(:,1:numPatches);
display_network(patches(:,randi(size(patches,2),200,1)),8);
```

```
%  Obtain random parameters theta
theta = initializeParameters(hiddenSize, visibleSize);

%%======================================================================
```



STEP 2: Implement sparseAutoencoderCost

```
You can implement all of the components (squared error cost, weight decay term,
sparsity penalty) in the cost function at once, but it may be easier to do
it step-by-step and run gradient checking (see STEP 3) after each step.  We
suggest implementing the sparseAutoencoderCost function using the following steps:

(a) Implement forward propagation in your neural network, and implement the
    squared error term of the cost function.  Implement backpropagation to
    compute the derivatives.   Then (using lambda=beta=0), run Gradient Checking
    to verify that the calculations corresponding to the squared error cost
    term are correct.

(b) Add in the weight decay term (in both the cost function and the derivative
    calculations), then re-run Gradient Checking to verify correctness.

(c) Add in the sparsity penalty term, then re-run Gradient Checking to
    verify correctness.

Feel free to change the training settings when debugging your
code.  (For example, reducing the training set size or
number of hidden units may make your code run faster; and setting beta
and/or lambda to zero may be helpful for debugging.)  However, in your
final submission of the visualized weights, please use parameters we
gave in Step 0 above.
```

```
[cost, grad] = sparseAutoencoderCost(theta, visibleSize, hiddenSize, lambda, ...
                                     sparsityParam, beta, patches);
```

```
%%======================================================================
% % %% STEP 3: Gradient Checking
% % %
% % % Hint: If you are debugging your code, performing gradient checking on smaller models
% % % and smaller training sets (e.g., using only 10 training examples and 1-2 hidden
% % % units) may speed things up.
% %
% % % First, lets make sure your numerical gradient computation is correct for a
% % % simple function.  After you have implemented computeNumericalGradient.m,
% % % run the following:
% checkNumericalGradient();
% %
% % % Now we can use it to check your cost function and derivative calculations
% % % for the sparse autoencoder.
% numgrad = computeNumericalGradient( @(x) sparseAutoencoderCost(x, visibleSize, ...
%                                                 hiddenSize, lambda, ...
%                                                 sparsityParam, beta, ...
%                                                 patches), theta);
%
% % Use this to visually compare the gradients side by side
% disp([numgrad grad]);
%
% % Compare numerically computed gradients with the ones obtained from backpropagation
% diff = norm(numgrad-grad)/norm(numgrad+grad);
% disp(diff); % Should be small. In our implementation, these values are
%             % usually less than 1e-9.
% if (diff > 1e-7)
%     fprintf('FIX GRAD IMPLEMENTATION\n');
%     return;
% end
%             % When you got this working, Congratulations!!!

%%======================================================================
```

STEP 4: After verifying that your implementation of

sparseAutoencoderCost is correct, You can start training your sparse autoencoder with minFunsc (L-BFGS).

```
%  Randomly initialize the parameters
theta = initializeParameters(hiddenSize, visibleSize);

%  Use minFunc to minimize the function
addpath minFunc/
options.Method = 'lbfgs'; % Here, we use L-BFGS to optimize our cost
                          % function. Generally, for minFunc to work, you
                          % need a function pointer with two outputs: the
                          % function value and the gradient. In our problem,
                          % sparseAutoencoderCost.m satisfies this.
options.maxIter = 400;    % Maximum number of iterations of L-BFGS to run
options.display = 'on';


[opttheta, cost] = minFunc( @(p) sparseAutoencoderCost(p, ...
                                   visibleSize, hiddenSize, ...
                                   lambda, sparsityParam, ...
                                   beta, patches), ...
                              theta, options);

%%======================================================================
```

Iteration	FunEvals	Step Length	Function Val	Opt Cond
1	5	3.21047e-02	1.13652e+02	8.97239e+03
2	6	1.00000e+00	9.23381e+01	4.52555e+03
3	7	1.00000e+00	8.40097e+01	2.07419e+03
4	8	1.00000e+00	7.93420e+01	1.95906e+03
5	9	1.00000e+00	7.11019e+01	2.58185e+03
6	10	1.00000e+00	5.44724e+01	2.74778e+03
7	11	1.00000e+00	3.41197e+01	6.18279e+02
8	12	1.00000e+00	3.22541e+01	7.18423e+02
9	13	1.00000e+00	3.12230e+01	5.88643e+02
10	14	1.00000e+00	3.02773e+01	3.75954e+02
11	15	1.00000e+00	2.97217e+01	2.26269e+02
12	16	1.00000e+00	2.95119e+01	2.21774e+02
13	17	1.00000e+00	2.94310e+01	1.65247e+02
14	18	1.00000e+00	2.92276e+01	2.22871e+02
15	19	1.00000e+00	2.89973e+01	3.27562e+02
16	20	1.00000e+00	2.84701e+01	4.62773e+02
17	22	4.21962e-01	2.82803e+01	4.44727e+02
18	23	1.00000e+00	2.79669e+01	3.29517e+02
19	24	1.00000e+00	2.77652e+01	1.80830e+02
20	25	1.00000e+00	2.76413e+01	2.41523e+02
21	26	1.00000e+00	2.74156e+01	2.99458e+02
22	27	1.00000e+00	2.68277e+01	4.20768e+02
23	28	1.00000e+00	2.60533e+01	3.53368e+02
24	29	1.00000e+00	2.52622e+01	2.55545e+02
25	30	1.00000e+00	2.49581e+01	3.43071e+02
26	31	1.00000e+00	2.46659e+01	2.44714e+02
27	32	1.00000e+00	2.44753e+01	2.16971e+02
28	33	1.00000e+00	2.39441e+01	2.98530e+02
29	34	1.00000e+00	2.29875e+01	3.52096e+02
30	36	3.40223e-01	2.25696e+01	3.57777e+02
31	37	1.00000e+00	2.20289e+01	3.24574e+02
32	38	1.00000e+00	2.17366e+01	3.11905e+02
33	39	1.00000e+00	2.13582e+01	2.12230e+02
34	40	1.00000e+00	2.11099e+01	2.22793e+02
35	41	1.00000e+00	2.08852e+01	2.43546e+02
36	42	1.00000e+00	2.06238e+01	2.77019e+02
37	43	1.00000e+00	2.02703e+01	2.63427e+02
38	44	1.00000e+00	1.97361e+01	2.33099e+02
39	45	1.00000e+00	1.93829e+01	2.43633e+02
40	46	1.00000e+00	1.91228e+01	1.58615e+02
41	47	1.00000e+00	1.90350e+01	1.41720e+02
42	48	1.00000e+00	1.88871e+01	1.50547e+02
43	49	1.00000e+00	1.87368e+01	1.86084e+02
44	50	1.00000e+00	1.85503e+01	1.71837e+02
45	51	1.00000e+00	1.83452e+01	1.55340e+02
46	52	1.00000e+00	1.81682e+01	1.69980e+02
47	53	1.00000e+00	1.79587e+01	1.60320e+02
48	54	1.00000e+00	1.76633e+01	1.69717e+02
49	55	1.00000e+00	1.75124e+01	1.78898e+02
50	56	1.00000e+00	1.73522e+01	1.28312e+02
51	57	1.00000e+00	1.72125e+01	1.13978e+02
52	58	1.00000e+00	1.71359e+01	1.24048e+02
53	59	1.00000e+00	1.70168e+01	1.26910e+02
54	60	1.00000e+00	1.69231e+01	1.43467e+02
55	61	1.00000e+00	1.67983e+01	1.06026e+02
56	62	1.00000e+00	1.67032e+01	1.02358e+02
57	63	1.00000e+00	1.65708e+01	1.15642e+02
58	64	1.00000e+00	1.64324e+01	1.83675e+02
59	65	1.00000e+00	1.62720e+01	1.12440e+02
60	66	1.00000e+00	1.61842e+01	9.74922e+01
61	67	1.00000e+00	1.60990e+01	1.14660e+02
62	68	1.00000e+00	1.60074e+01	1.41495e+02

63	69	1.00000e+00	1.58979e+01	1.29284e+02
64	70	1.00000e+00	1.57921e+01	1.02429e+02
65	71	1.00000e+00	1.56878e+01	9.86337e+01
66	72	1.00000e+00	1.55982e+01	1.14784e+02
67	73	1.00000e+00	1.55007e+01	1.17310e+02
68	74	1.00000e+00	1.54185e+01	9.25934e+01
69	75	1.00000e+00	1.53539e+01	8.84370e+01
70	76	1.00000e+00	1.52940e+01	9.24711e+01
71	77	1.00000e+00	1.52027e+01	9.98076e+01
72	78	1.00000e+00	1.50940e+01	9.67125e+01
73	79	1.00000e+00	1.50248e+01	7.99907e+01
74	80	1.00000e+00	1.49638e+01	7.28227e+01
75	81	1.00000e+00	1.48921e+01	7.70369e+01
76	82	1.00000e+00	1.48322e+01	9.07542e+01
77	83	1.00000e+00	1.47814e+01	6.49443e+01
78	84	1.00000e+00	1.47465e+01	6.12188e+01
79	85	1.00000e+00	1.47077e+01	7.17904e+01
80	86	1.00000e+00	1.46486e+01	8.10172e+01
81	87	1.00000e+00	1.45849e+01	9.44481e+01
82	88	1.00000e+00	1.45319e+01	8.38534e+01
83	89	1.00000e+00	1.44735e+01	6.93396e+01
84	90	1.00000e+00	1.44248e+01	7.59434e+01
85	91	1.00000e+00	1.43847e+01	7.04515e+01
86	92	1.00000e+00	1.43279e+01	9.10882e+01
87	93	1.00000e+00	1.42921e+01	7.26387e+01
88	94	1.00000e+00	1.42675e+01	5.49394e+01
89	95	1.00000e+00	1.42376e+01	5.27128e+01
90	96	1.00000e+00	1.42072e+01	6.19149e+01
91	97	1.00000e+00	1.41681e+01	7.22273e+01
92	98	1.00000e+00	1.41173e+01	6.47304e+01
93	99	1.00000e+00	1.40767e+01	5.78533e+01
94	100	1.00000e+00	1.40518e+01	5.35331e+01
95	101	1.00000e+00	1.40211e+01	6.10700e+01
96	102	1.00000e+00	1.39948e+01	5.30652e+01
97	103	1.00000e+00	1.39713e+01	5.33685e+01
98	104	1.00000e+00	1.39370e+01	5.56856e+01
99	105	1.00000e+00	1.38998e+01	6.29702e+01
100	106	1.00000e+00	1.38675e+01	6.84580e+01
101	107	1.00000e+00	1.38421e+01	5.00109e+01
102	108	1.00000e+00	1.38257e+01	4.90808e+01
103	109	1.00000e+00	1.38104e+01	4.75572e+01
104	110	1.00000e+00	1.37828e+01	5.21168e+01
105	111	1.00000e+00	1.37628e+01	6.07072e+01
106	112	1.00000e+00	1.37460e+01	5.64902e+01
107	113	1.00000e+00	1.37206e+01	6.14551e+01
108	114	1.00000e+00	1.36892e+01	6.21192e+01
109	115	1.00000e+00	1.36608e+01	5.92560e+01
110	116	1.00000e+00	1.36503e+01	5.94608e+01
111	117	1.00000e+00	1.36324e+01	4.46781e+01
112	118	1.00000e+00	1.36245e+01	3.58773e+01
113	119	1.00000e+00	1.36146e+01	3.79240e+01
114	120	1.00000e+00	1.35867e+01	5.50666e+01
115	121	1.00000e+00	1.35636e+01	5.31009e+01
116	122	1.00000e+00	1.35309e+01	5.53064e+01
117	123	1.00000e+00	1.35043e+01	5.41726e+01
118	124	1.00000e+00	1.34911e+01	3.58434e+01
119	125	1.00000e+00	1.34838e+01	3.67773e+01
120	126	1.00000e+00	1.34691e+01	4.90419e+01
121	127	1.00000e+00	1.34417e+01	6.62580e+01
122	128	1.00000e+00	1.34051e+01	6.39655e+01
123	129	1.00000e+00	1.33865e+01	5.81627e+01
124	130	1.00000e+00	1.33747e+01	3.63962e+01
125	132	4.47323e-01	1.33693e+01	3.63287e+01
126	133	1.00000e+00	1.33612e+01	3.52770e+01

127	134	1.00000e+00	1.33443e+01	4.08288e+01
128	135	1.00000e+00	1.33229e+01	4.74229e+01
129	136	1.00000e+00	1.32964e+01	5.85758e+01
130	137	1.00000e+00	1.32758e+01	6.07799e+01
131	138	1.00000e+00	1.32583e+01	3.62751e+01
132	139	1.00000e+00	1.32555e+01	4.30597e+01
133	140	1.00000e+00	1.32459e+01	3.56932e+01
134	141	1.00000e+00	1.32409e+01	2.94076e+01
135	142	1.00000e+00	1.32236e+01	3.63305e+01
136	143	1.00000e+00	1.31949e+01	5.39484e+01
137	144	1.00000e+00	1.31746e+01	5.93188e+01
138	145	1.00000e+00	1.31588e+01	4.62077e+01
139	146	1.00000e+00	1.31475e+01	3.03545e+01
140	147	1.00000e+00	1.31379e+01	3.49262e+01
141	148	1.00000e+00	1.31268e+01	4.29428e+01
142	149	1.00000e+00	1.31078e+01	5.36767e+01
143	150	1.00000e+00	1.30916e+01	5.31026e+01
144	151	1.00000e+00	1.30771e+01	3.69503e+01
145	152	1.00000e+00	1.30604e+01	3.86932e+01
146	153	1.00000e+00	1.30498e+01	4.03859e+01
147	154	1.00000e+00	1.30310e+01	4.54486e+01
148	155	1.00000e+00	1.30176e+01	4.40986e+01
149	156	1.00000e+00	1.30070e+01	3.45953e+01
150	157	1.00000e+00	1.29948e+01	3.06902e+01
151	158	1.00000e+00	1.29861e+01	3.46159e+01
152	159	1.00000e+00	1.29702e+01	3.70364e+01
153	160	1.00000e+00	1.29557e+01	6.02585e+01
154	161	1.00000e+00	1.29354e+01	3.86317e+01
155	162	1.00000e+00	1.29228e+01	2.83996e+01
156	163	1.00000e+00	1.29146e+01	2.65523e+01
157	164	1.00000e+00	1.29062e+01	3.89611e+01
158	165	1.00000e+00	1.28955e+01	3.33682e+01
159	166	1.00000e+00	1.28790e+01	3.46926e+01
160	167	1.00000e+00	1.28664e+01	4.39571e+01
161	168	1.00000e+00	1.28520e+01	3.36504e+01
162	169	1.00000e+00	1.28389e+01	3.06674e+01
163	170	1.00000e+00	1.28291e+01	3.59178e+01
164	171	1.00000e+00	1.28218e+01	3.06742e+01
165	172	1.00000e+00	1.28146e+01	2.80330e+01
166	173	1.00000e+00	1.28010e+01	3.93833e+01
167	174	1.00000e+00	1.27892e+01	4.71775e+01
168	175	1.00000e+00	1.27728e+01	4.26115e+01
169	176	1.00000e+00	1.27620e+01	4.11321e+01
170	177	1.00000e+00	1.27528e+01	2.64044e+01
171	178	1.00000e+00	1.27472e+01	2.95631e+01
172	179	1.00000e+00	1.27401e+01	3.04626e+01
173	180	1.00000e+00	1.27358e+01	5.67096e+01
174	181	1.00000e+00	1.27221e+01	2.75111e+01
175	182	1.00000e+00	1.27139e+01	2.44526e+01
176	183	1.00000e+00	1.27041e+01	3.53060e+01
177	184	1.00000e+00	1.26934e+01	5.07367e+01
178	185	1.00000e+00	1.26818e+01	3.79753e+01
179	186	1.00000e+00	1.26729e+01	2.44100e+01
180	187	1.00000e+00	1.26658e+01	2.77649e+01
181	188	1.00000e+00	1.26613e+01	3.19950e+01
182	189	1.00000e+00	1.26501e+01	3.67489e+01
183	190	1.00000e+00	1.26396e+01	5.06816e+01
184	191	1.00000e+00	1.26258e+01	3.05538e+01
185	192	1.00000e+00	1.26167e+01	2.55378e+01
186	193	1.00000e+00	1.26101e+01	2.78906e+01
187	194	1.00000e+00	1.26053e+01	3.59085e+01
188	195	1.00000e+00	1.25987e+01	2.72515e+01
189	196	1.00000e+00	1.25877e+01	2.62341e+01
190	197	1.00000e+00	1.25797e+01	3.13209e+01

191	198	1.00000e+00	1.25729e+01	4.40229e+01
192	199	1.00000e+00	1.25644e+01	2.93733e+01
193	200	1.00000e+00	1.25570e+01	2.36948e+01
194	201	1.00000e+00	1.25503e+01	2.59661e+01
195	202	1.00000e+00	1.25452e+01	2.92970e+01
196	203	1.00000e+00	1.25393e+01	2.24859e+01
197	204	1.00000e+00	1.25326e+01	2.30172e+01
198	205	1.00000e+00	1.25259e+01	2.92254e+01
199	206	1.00000e+00	1.25189e+01	2.84730e+01
200	207	1.00000e+00	1.25122e+01	2.20450e+01
201	208	1.00000e+00	1.25035e+01	3.13024e+01
202	209	1.00000e+00	1.24979e+01	3.22839e+01
203	210	1.00000e+00	1.24917e+01	2.53920e+01
204	211	1.00000e+00	1.24858e+01	2.71311e+01
205	212	1.00000e+00	1.24824e+01	2.51908e+01
206	213	1.00000e+00	1.24795e+01	2.49228e+01
207	214	1.00000e+00	1.24682e+01	3.47505e+01
208	215	1.00000e+00	1.24645e+01	4.16626e+01
209	216	1.00000e+00	1.24563e+01	2.21868e+01
210	217	1.00000e+00	1.24522e+01	1.64754e+01
211	218	1.00000e+00	1.24489e+01	1.97361e+01
212	219	1.00000e+00	1.24443e+01	2.44888e+01
213	220	1.00000e+00	1.24384e+01	2.72656e+01
214	221	1.00000e+00	1.24323e+01	2.29048e+01
215	222	1.00000e+00	1.24255e+01	2.47856e+01
216	223	1.00000e+00	1.24217e+01	2.44447e+01
217	224	1.00000e+00	1.24170e+01	2.30470e+01
218	225	1.00000e+00	1.24075e+01	2.65234e+01
219	227	4.50888e-01	1.24045e+01	2.14437e+01
220	228	1.00000e+00	1.24023e+01	1.66599e+01
221	229	1.00000e+00	1.23987e+01	1.88663e+01
222	230	1.00000e+00	1.23946e+01	2.36747e+01
223	231	1.00000e+00	1.23871e+01	2.93953e+01
224	232	1.00000e+00	1.23822e+01	3.10787e+01
225	233	1.00000e+00	1.23774e+01	1.79711e+01
226	234	1.00000e+00	1.23746e+01	1.48433e+01
227	235	1.00000e+00	1.23731e+01	1.62508e+01
228	236	1.00000e+00	1.23683e+01	2.17311e+01
229	237	1.00000e+00	1.23640e+01	2.76543e+01
230	238	1.00000e+00	1.23593e+01	2.02299e+01
231	239	1.00000e+00	1.23531e+01	1.88890e+01
232	240	1.00000e+00	1.23491e+01	2.06812e+01
233	241	1.00000e+00	1.23464e+01	3.10339e+01
234	242	1.00000e+00	1.23420e+01	1.51291e+01
235	243	1.00000e+00	1.23401e+01	1.41926e+01
236	244	1.00000e+00	1.23375e+01	1.74512e+01
237	245	1.00000e+00	1.23340e+01	2.04557e+01
238	246	1.00000e+00	1.23287e+01	2.11937e+01
239	247	1.00000e+00	1.23236e+01	2.07157e+01
240	248	1.00000e+00	1.23203e+01	1.97145e+01
241	249	1.00000e+00	1.23181e+01	1.55082e+01
242	250	1.00000e+00	1.23157e+01	1.57530e+01
243	251	1.00000e+00	1.23115e+01	1.92052e+01
244	252	1.00000e+00	1.23079e+01	2.84249e+01
245	253	1.00000e+00	1.23039e+01	1.94732e+01
246	254	1.00000e+00	1.23003e+01	1.55967e+01
247	255	1.00000e+00	1.22980e+01	1.64483e+01
248	256	1.00000e+00	1.22956e+01	2.19404e+01
249	257	1.00000e+00	1.22931e+01	1.71722e+01
250	258	1.00000e+00	1.22901e+01	1.78595e+01
251	259	1.00000e+00	1.22865e+01	2.04447e+01
252	260	1.00000e+00	1.22832e+01	2.55384e+01
253	261	1.00000e+00	1.22797e+01	1.71978e+01
254	262	1.00000e+00	1.22778e+01	1.34952e+01

255	263	1.00000e+00	1.22762e+01	1.49632e+01
256	264	1.00000e+00	1.22740e+01	1.86988e+01
257	265	1.00000e+00	1.22710e+01	2.03146e+01
258	266	1.00000e+00	1.22683e+01	1.71579e+01
259	267	1.00000e+00	1.22658e+01	1.42244e+01
260	268	1.00000e+00	1.22633e+01	1.79203e+01
261	269	1.00000e+00	1.22601e+01	2.06162e+01
262	270	1.00000e+00	1.22574e+01	2.44104e+01
263	271	1.00000e+00	1.22545e+01	1.42845e+01
264	272	1.00000e+00	1.22531e+01	1.30179e+01
265	273	1.00000e+00	1.22518e+01	1.49086e+01
266	274	1.00000e+00	1.22499e+01	1.74518e+01
267	275	1.00000e+00	1.22476e+01	2.00141e+01
268	276	1.00000e+00	1.22452e+01	1.59621e+01
269	277	1.00000e+00	1.22429e+01	1.51471e+01
270	278	1.00000e+00	1.22407e+01	1.73109e+01
271	279	1.00000e+00	1.22382e+01	1.88891e+01
272	280	1.00000e+00	1.22359e+01	1.78951e+01
273	281	1.00000e+00	1.22339e+01	1.33562e+01
274	282	1.00000e+00	1.22321e+01	1.27351e+01
275	283	1.00000e+00	1.22307e+01	1.43380e+01
276	284	1.00000e+00	1.22284e+01	1.63047e+01
277	285	1.00000e+00	1.22261e+01	1.86151e+01
278	286	1.00000e+00	1.22237e+01	1.38660e+01
279	287	1.00000e+00	1.22216e+01	1.09184e+01
280	288	1.00000e+00	1.22206e+01	1.14070e+01
281	289	1.00000e+00	1.22190e+01	1.25611e+01
282	290	1.00000e+00	1.22169e+01	1.58419e+01
283	291	1.00000e+00	1.22147e+01	1.32252e+01
284	292	1.00000e+00	1.22130e+01	1.12240e+01
285	293	1.00000e+00	1.22119e+01	1.15773e+01
286	294	1.00000e+00	1.22107e+01	1.21774e+01
287	295	1.00000e+00	1.22082e+01	1.39018e+01
288	296	1.00000e+00	1.22071e+01	2.61584e+01
289	297	1.00000e+00	1.22042e+01	1.34474e+01
290	298	1.00000e+00	1.22030e+01	1.04180e+01
291	299	1.00000e+00	1.22017e+01	1.13271e+01
292	300	1.00000e+00	1.22005e+01	1.40861e+01
293	301	1.00000e+00	1.21988e+01	1.24429e+01
294	302	1.00000e+00	1.21974e+01	1.03375e+01
295	303	1.00000e+00	1.21960e+01	1.32180e+01
296	304	1.00000e+00	1.21947e+01	1.43803e+01
297	305	1.00000e+00	1.21929e+01	1.57761e+01
298	306	1.00000e+00	1.21907e+01	1.75528e+01
299	307	1.00000e+00	1.21891e+01	1.48754e+01
300	308	1.00000e+00	1.21880e+01	1.06899e+01
301	309	1.00000e+00	1.21871e+01	1.06678e+01
302	310	1.00000e+00	1.21864e+01	1.23615e+01
303	311	1.00000e+00	1.21847e+01	1.52095e+01
304	312	1.00000e+00	1.21833e+01	1.87173e+01
305	313	1.00000e+00	1.21816e+01	1.25510e+01
306	314	1.00000e+00	1.21803e+01	9.99407e+00
307	315	1.00000e+00	1.21794e+01	1.10742e+01
308	316	1.00000e+00	1.21781e+01	1.19963e+01
309	318	4.49778e-01	1.21773e+01	1.38476e+01
310	319	1.00000e+00	1.21762e+01	9.66053e+00
311	320	1.00000e+00	1.21753e+01	8.74540e+00
312	321	1.00000e+00	1.21742e+01	1.20256e+01
313	322	1.00000e+00	1.21726e+01	1.57678e+01
314	323	1.00000e+00	1.21712e+01	1.84404e+01
315	324	1.00000e+00	1.21697e+01	1.10555e+01
316	325	1.00000e+00	1.21688e+01	8.31391e+00
317	326	1.00000e+00	1.21683e+01	9.50363e+00
318	327	1.00000e+00	1.21673e+01	1.18545e+01

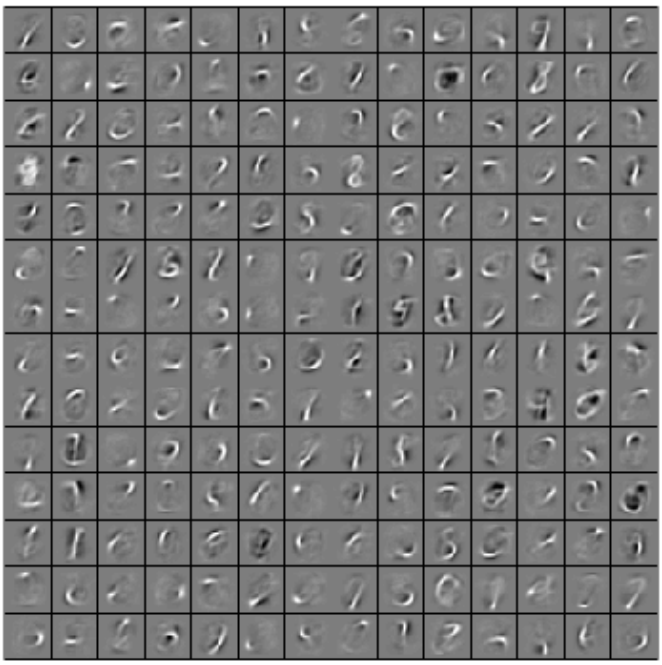
319	328	1.00000e+00	1.21661e+01	1.43088e+01
320	329	1.00000e+00	1.21648e+01	1.16719e+01
321	330	1.00000e+00	1.21635e+01	9.79770e+00
322	331	1.00000e+00	1.21626e+01	1.06386e+01
323	332	1.00000e+00	1.21618e+01	1.05614e+01
324	333	1.00000e+00	1.21607e+01	1.19534e+01
325	334	1.00000e+00	1.21599e+01	1.04095e+01
326	335	1.00000e+00	1.21592e+01	9.93615e+00
327	336	1.00000e+00	1.21577e+01	1.12130e+01
328	337	1.00000e+00	1.21569e+01	1.17342e+01
329	338	1.00000e+00	1.21560e+01	1.04231e+01
330	339	1.00000e+00	1.21552e+01	1.04280e+01
331	340	1.00000e+00	1.21544e+01	1.17533e+01
332	341	1.00000e+00	1.21532e+01	1.23719e+01
333	342	1.00000e+00	1.21522e+01	1.52788e+01
334	343	1.00000e+00	1.21512e+01	8.51478e+00
335	344	1.00000e+00	1.21508e+01	7.32403e+00
336	345	1.00000e+00	1.21504e+01	8.89511e+00
337	346	1.00000e+00	1.21496e+01	1.10915e+01
338	347	1.00000e+00	1.21482e+01	1.35331e+01
339	348	1.00000e+00	1.21476e+01	1.69664e+01
340	349	1.00000e+00	1.21463e+01	8.89662e+00
341	350	1.00000e+00	1.21458e+01	6.39624e+00
342	351	1.00000e+00	1.21456e+01	7.93032e+00
343	352	1.00000e+00	1.21450e+01	9.87939e+00
344	353	1.00000e+00	1.21440e+01	1.14345e+01
345	354	1.00000e+00	1.21430e+01	1.38395e+01
346	355	1.00000e+00	1.21419e+01	1.03441e+01
347	356	1.00000e+00	1.21411e+01	9.49758e+00
348	357	1.00000e+00	1.21406e+01	9.41771e+00
349	358	1.00000e+00	1.21400e+01	9.23909e+00
350	359	1.00000e+00	1.21393e+01	9.23020e+00
351	360	1.00000e+00	1.21387e+01	8.38601e+00
352	361	1.00000e+00	1.21379e+01	9.95868e+00
353	362	1.00000e+00	1.21372e+01	1.04609e+01
354	363	1.00000e+00	1.21362e+01	1.27120e+01
355	364	1.00000e+00	1.21353e+01	7.98194e+00
356	365	1.00000e+00	1.21346e+01	6.80297e+00
357	366	1.00000e+00	1.21342e+01	7.06019e+00
358	367	1.00000e+00	1.21337e+01	9.79474e+00
359	368	1.00000e+00	1.21330e+01	7.81874e+00
360	369	1.00000e+00	1.21321e+01	8.47964e+00
361	370	1.00000e+00	1.21314e+01	1.03494e+01
362	371	1.00000e+00	1.21308e+01	7.99121e+00
363	372	1.00000e+00	1.21301e+01	6.91529e+00
364	373	1.00000e+00	1.21294e+01	8.22613e+00
365	374	1.00000e+00	1.21287e+01	1.05689e+01
366	375	1.00000e+00	1.21279e+01	9.12317e+00
367	376	1.00000e+00	1.21274e+01	6.99397e+00
368	377	1.00000e+00	1.21270e+01	6.37910e+00
369	378	1.00000e+00	1.21266e+01	7.19685e+00
370	379	1.00000e+00	1.21258e+01	9.12229e+00
371	380	1.00000e+00	1.21250e+01	1.07647e+01
372	381	1.00000e+00	1.21243e+01	9.64005e+00
373	382	1.00000e+00	1.21237e+01	6.80747e+00
374	383	1.00000e+00	1.21233e+01	5.98637e+00
375	384	1.00000e+00	1.21228e+01	7.59507e+00
376	385	1.00000e+00	1.21221e+01	9.64548e+00
377	386	1.00000e+00	1.21215e+01	1.08134e+01
378	387	1.00000e+00	1.21209e+01	7.69920e+00
379	388	1.00000e+00	1.21204e+01	6.56536e+00
380	389	1.00000e+00	1.21199e+01	7.18346e+00
381	390	1.00000e+00	1.21194e+01	1.11710e+01
382	391	1.00000e+00	1.21189e+01	7.19083e+00

```
         383         392     1.00000e+00     1.21186e+01     6.36481e+00
         384         393     1.00000e+00     1.21182e+01     7.32874e+00
         385         394     1.00000e+00     1.21175e+01     9.04960e+00
         386         395     1.00000e+00     1.21170e+01     1.13982e+01
         387         396     1.00000e+00     1.21163e+01     7.05622e+00
         388         397     1.00000e+00     1.21160e+01     6.29492e+00
         389         398     1.00000e+00     1.21156e+01     7.38628e+00
         390         399     1.00000e+00     1.21151e+01     9.27573e+00
         391         400     1.00000e+00     1.21144e+01     1.01420e+01
         392         401     1.00000e+00     1.21137e+01     8.17526e+00
         393         402     1.00000e+00     1.21132e+01     6.58309e+00
         394         403     1.00000e+00     1.21129e+01     7.07284e+00
         395         404     1.00000e+00     1.21126e+01     8.16265e+00
         396         405     1.00000e+00     1.21119e+01     9.11383e+00
         397         406     1.00000e+00     1.21114e+01     9.88862e+00
         398         407     1.00000e+00     1.21110e+01     6.72269e+00
         399         408     1.00000e+00     1.21106e+01     6.59413e+00
         400         409     1.00000e+00     1.21103e+01     7.18950e+00
Exceeded Maximum Number of Iterations
```

STEP 5: Visualization

```
W1 = reshape(opttheta(1:hiddenSize*visibleSize), hiddenSize, visibleSize);
display_network(W1', 12);

print -djpeg weights.jpg   % save the visualization to a file
```



Published with MATLAB® R2016b

Contents

CS294A/CS294W Softmax Exercise

```
%  Instructions
%  ------------
%
%  This file contains code that helps you get started on the
%  softmax exercise. You will need to write the softmax cost function
%  in softmaxCost.m and the softmax prediction function in softmaxPred.m.
%  For this exercise, you will not need to change any code in this file,
%  or any other files other than those mentioned above.
%  (However, you may be required to do so in later exercises)

%%======================================================================
```

STEP 0: Initialise constants and parameters

Here we define and initialise some constants which allow your code
to be used more generally on any arbitrary input.
We also initialise some parameters used for tuning the model.

```
inputSize = 28 * 28; % Size of input vector (MNIST images are 28x28)
numClasses = 10;     % Number of classes (MNIST images fall into 10 classes)

lambda = 1e-4; % Weight decay parameter

%%======================================================================
```

STEP 1: Load data

In this section, we load the input and output data.
For softmax regression on MNIST pixels,
the input data is the images, and
the output data is the labels.

```
% Change the filenames if you've saved the files under different names
% On some platforms, the files might be saved as
% train-images.idx3-ubyte / train-labels.idx1-ubyte

images = loadMNISTImages('train-images-idx3-ubyte');
labels = loadMNISTLabels('train-labels-idx1-ubyte');
labels(labels==0) = 10; % Remap 0 to 10

inputData = images;
```

```
% For debugging purposes, you may wish to reduce the size of the input data
% in order to speed up gradient checking.
% Here, we create synthetic dataset using random data for testing

DEBUG = false; % Set DEBUG to true when debugging.
if DEBUG
    inputSize = 8;
    inputData = randn(8, 100);
    labels = randi(10, 100, 1);
end

% Randomly initialise theta
theta = 0.005 * randn(numClasses * inputSize, 1);

%%======================================================================
```

STEP 2: Implement softmaxCost

Implement softmaxCost in softmaxCost.m.

```
[cost, grad] = softmaxCost(theta, numClasses, inputSize, lambda, inputData, labels);

%%======================================================================
```

STEP 3: Gradient checking

As with any learning algorithm, you should always check that your
gradients are correct before learning the parameters.

```
if DEBUG
    numGrad = computeNumericalGradient( @(x) softmaxCost(x, numClasses, ...
                                    inputSize, lambda, inputData, labels), theta);

    % Use this to visually compare the gradients side by side
    disp([numGrad grad]);

    % Compare numerically computed gradients with those computed analytically
    diff = norm(numGrad-grad)/norm(numGrad+grad);
    disp(diff);
    % The difference should be small.
    % In our implementation, these values are usually less than 1e-7.

    % When your gradients are correct, congratulations!
end

%%======================================================================
```

STEP 4: Learning parameters

Once you have verified that your gradients are correct,
you can start training your softmax regression code using softmaxTrain
(which uses minFunc).

```
options.maxIter = 100;
softmaxModel = softmaxTrain(inputSize, numClasses, lambda, ...
```

```
                            inputData, labels, options);

% Although we only use 100 iterations here to train a classifier for the
% MNIST data set, in practice, training for more iterations is usually
% beneficial.

%%======================================================================
```

Iteration	FunEvals	Step Length	Function Val	Opt Cond
1	3	1.92328e-01	2.10163e+00	4.70656e+01
2	4	1.00000e+00	7.85183e-01	2.66556e+01
3	6	2.46952e-01	6.61912e-01	1.44679e+01
4	7	1.00000e+00	6.08051e-01	8.56711e+00
5	8	1.00000e+00	5.36527e-01	6.62465e+00
6	9	1.00000e+00	5.15677e-01	1.72315e+01
7	10	1.00000e+00	4.62435e-01	6.12149e+00
8	11	1.00000e+00	4.48904e-01	3.76580e+00
9	12	1.00000e+00	4.34593e-01	3.69978e+00
10	13	1.00000e+00	4.16014e-01	3.70404e+00
11	14	1.00000e+00	3.94745e-01	6.71925e+00
12	15	1.00000e+00	3.70423e-01	3.53918e+00
13	16	1.00000e+00	3.60588e-01	2.49192e+00
14	17	1.00000e+00	3.50762e-01	2.83137e+00
15	18	1.00000e+00	3.43581e-01	2.37286e+00
16	19	1.00000e+00	3.32071e-01	2.05780e+00
17	20	1.00000e+00	3.25905e-01	2.28273e+00
18	21	1.00000e+00	3.18073e-01	2.39495e+00
19	22	1.00000e+00	3.12691e-01	2.01914e+00
20	23	1.00000e+00	3.09198e-01	1.21409e+00
21	24	1.00000e+00	3.05992e-01	1.23137e+00
22	25	1.00000e+00	3.02965e-01	1.42426e+00
23	26	1.00000e+00	2.98200e-01	1.31894e+00
24	27	1.00000e+00	2.97213e-01	3.35947e+00
25	28	1.00000e+00	2.91868e-01	1.18804e+00
26	29	1.00000e+00	2.90237e-01	7.82587e-01
27	30	1.00000e+00	2.88226e-01	8.32320e-01
28	31	1.00000e+00	2.85768e-01	8.45418e-01
29	32	1.00000e+00	2.81360e-01	1.08530e+00
30	33	1.00000e+00	2.79893e-01	1.74866e+00
31	34	1.00000e+00	2.76970e-01	6.45392e-01
32	35	1.00000e+00	2.75917e-01	5.87191e-01
33	36	1.00000e+00	2.74625e-01	7.40947e-01
34	37	1.00000e+00	2.72631e-01	7.82949e-01
35	38	1.00000e+00	2.69467e-01	7.41057e-01
36	39	1.00000e+00	2.67873e-01	1.27439e+00
37	40	1.00000e+00	2.66345e-01	4.90607e-01
38	41	1.00000e+00	2.65709e-01	4.59497e-01
39	42	1.00000e+00	2.64964e-01	5.99799e-01
40	43	1.00000e+00	2.63623e-01	6.68762e-01
41	44	1.00000e+00	2.61392e-01	6.22463e-01
42	45	1.00000e+00	2.59557e-01	8.16274e-01
43	46	1.00000e+00	2.57570e-01	4.07482e-01
44	47	1.00000e+00	2.56606e-01	3.43760e-01
45	48	1.00000e+00	2.55567e-01	3.73585e-01
46	49	1.00000e+00	2.54391e-01	5.03553e-01
47	50	1.00000e+00	2.53101e-01	3.62406e-01
48	51	1.00000e+00	2.51938e-01	3.49691e-01
49	52	1.00000e+00	2.50949e-01	4.27099e-01
50	53	1.00000e+00	2.50125e-01	3.05966e-01

```
         51          54     1.00000e+00     2.49128e-01     2.83601e-01
         52          55     1.00000e+00     2.48276e-01     3.63655e-01
         53          56     1.00000e+00     2.47403e-01     3.27255e-01
         54          57     1.00000e+00     2.46714e-01     2.67334e-01
         55          58     1.00000e+00     2.45895e-01     2.68427e-01
         56          59     1.00000e+00     2.45052e-01     3.17958e-01
         57          60     1.00000e+00     2.44200e-01     3.29978e-01
         58          61     1.00000e+00     2.43690e-01     2.27922e-01
         59          62     1.00000e+00     2.43237e-01     2.56569e-01
         60          63     1.00000e+00     2.42689e-01     2.71644e-01
         61          64     1.00000e+00     2.41635e-01     2.85248e-01
         62          66     4.45067e-01     2.41109e-01     3.82220e-01
         63          67     1.00000e+00     2.40406e-01     2.30963e-01
         64          68     1.00000e+00     2.39841e-01     1.91986e-01
         65          69     1.00000e+00     2.39184e-01     2.04713e-01
         66          70     1.00000e+00     2.38559e-01     2.61557e-01
         67          71     1.00000e+00     2.38063e-01     1.81270e-01
         68          72     1.00000e+00     2.37640e-01     1.83703e-01
         69          73     1.00000e+00     2.37029e-01     1.95719e-01
         70          74     1.00000e+00     2.36596e-01     4.21623e-01
         71          75     1.00000e+00     2.35902e-01     1.80704e-01
         72          76     1.00000e+00     2.35601e-01     1.42285e-01
         73          77     1.00000e+00     2.35240e-01     1.64763e-01
         74          78     1.00000e+00     2.34648e-01     1.81349e-01
         75          79     1.00000e+00     2.34201e-01     3.85022e-01
         76          80     1.00000e+00     2.33607e-01     1.52380e-01
         77          81     1.00000e+00     2.33364e-01     1.27165e-01
         78          82     1.00000e+00     2.33070e-01     1.46910e-01
         79          83     1.00000e+00     2.32519e-01     1.56313e-01
         80          84     1.00000e+00     2.31882e-01     2.97942e-01
         81          85     1.00000e+00     2.31298e-01     1.34825e-01
         82          86     1.00000e+00     2.31033e-01     1.10926e-01
         83          87     1.00000e+00     2.30692e-01     1.35141e-01
         84          88     1.00000e+00     2.30380e-01     2.66073e-01
         85          89     1.00000e+00     2.29984e-01     1.45046e-01
         86          90     1.00000e+00     2.29609e-01     1.17860e-01
         87          91     1.00000e+00     2.29356e-01     1.21363e-01
         88          92     1.00000e+00     2.28836e-01     1.32991e-01
         89          94     4.54307e-01     2.28588e-01     1.82409e-01
         90          95     1.00000e+00     2.28294e-01     1.09993e-01
         91          96     1.00000e+00     2.28057e-01     1.02678e-01
         92          97     1.00000e+00     2.27801e-01     1.36576e-01
         93          98     1.00000e+00     2.27520e-01     1.24526e-01
         94          99     1.00000e+00     2.27253e-01     1.20848e-01
         95         100     1.00000e+00     2.26973e-01     1.16590e-01
         96         101     1.00000e+00     2.26746e-01     1.28364e-01
         97         102     1.00000e+00     2.26506e-01     1.17284e-01
         98         103     1.00000e+00     2.26247e-01     1.19643e-01
         99         104     1.00000e+00     2.26005e-01     1.51340e-01
        100         105     1.00000e+00     2.25777e-01     1.23834e-01
Exceeded Maximum Number of Iterations
```

STEP 5: Testing

```
You should now test your model against the test images.
To do this, you will first need to write softmaxPredict
(in softmaxPredict.m), which should return predictions
given a softmax model and the input data.
```

```
images = loadMNISTImages('t10k-images-idx3-ubyte');
labels = loadMNISTLabels('t10k-labels-idx1-ubyte');
labels(labels==0) = 10; % Remap 0 to 10

inputData = images;

% You will have to implement softmaxPredict in softmaxPredict.m
[pred] = softmaxPredict(softmaxModel, inputData);

acc = mean(labels(:) == pred(:));
fprintf('Accuracy: %0.3f%%\n', acc * 100);

% Accuracy is the proportion of correctly classified images
% After 100 iterations, the results for our implementation were:
%
% Accuracy: 92.200%
%
% If your values are too low (accuracy less than 0.91), you should check
% your code for errors, and make sure you are training on the
% entire data set of 60000 28x28 training images
% (unless you modified the loading code, this should be the case)
```

```
Accuracy: 92.220%
```

Published with MATLAB® R2016b

Contents

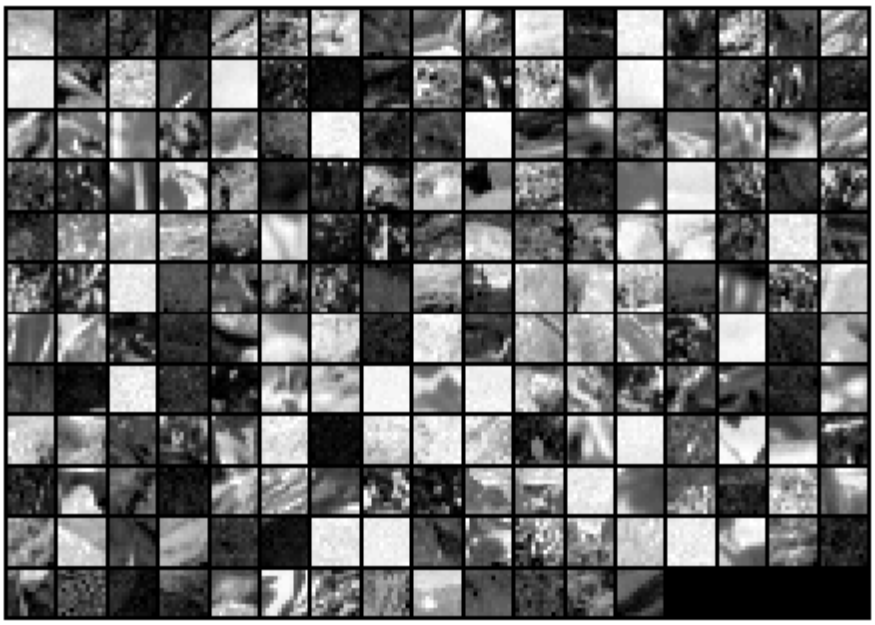

```
%%================================================================
```

Step 0a: Load data

Here we provide the code to load natural image data into x.
x will be a 144 * 10000 matrix, where the kth column x(:, k) corresponds to
the raw image data from the kth 12x12 image patch sampled.
You do not need to change the code below.

```
close all
x = sampleIMAGESRAW();
figure('name','Raw images');
randsel = randi(size(x,2),200,1); % A random selection of samples for visualization
display_network(x(:,randsel));

%%================================================================
```



Step 0b: Zero-mean the data (by row)

You can make use of the mean and repmat/bsxfun functions.

```
% --------------------- YOUR CODE HERE --------------------
x = x - repmat(mean(x,1), size(x,1),1);

%%=================================================================
```

Step 1a: Implement PCA to obtain xRot

Implement PCA to obtain xRot, the matrix in which the data is expressed
with respect to the eigenbasis of sigma, which is the matrix U.

```
% --------------------- YOUR CODE HERE --------------------
xRot = zeros(size(x)); % You need to compute this
sigma = x * x' / size(x, 2);
[U,S,V] = svd(sigma);

xRot(:,:) = U' * x;     % rotated version of the data.

%%=================================================================
```

Step 1b: Check your implementation of PCA

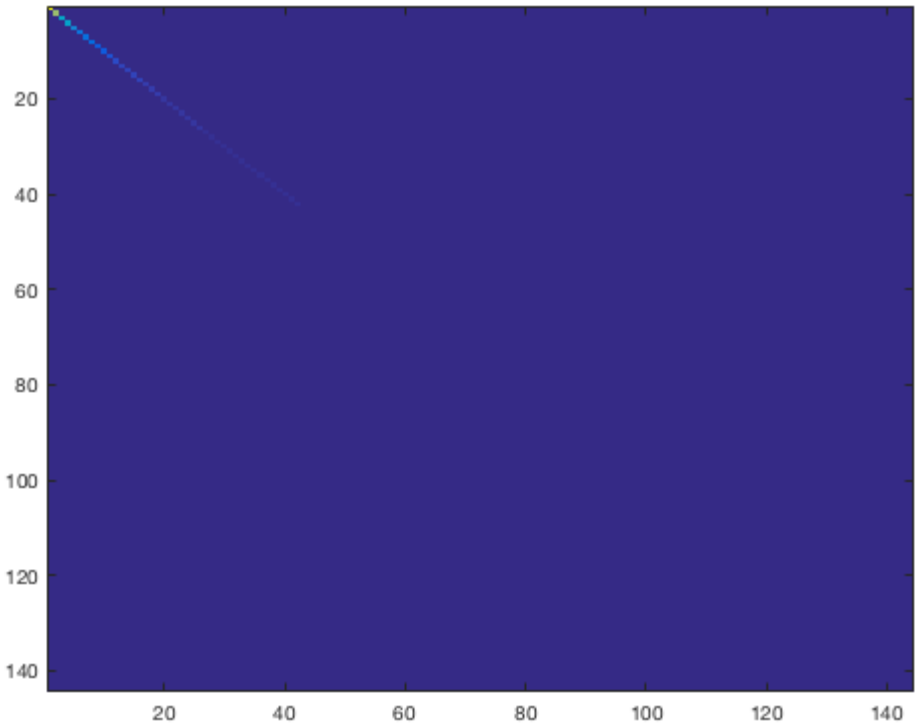
The covariance matrix for the data expressed with respect to the basis U
should be a diagonal matrix with non-zero entries only along the main
diagonal. We will verify this here.
Write code to compute the covariance matrix, covar.

When visualised as an image, you should see a straight line across the diagonal (non-zero entries) against a blue background (zero entries).

```
% --------------------- YOUR CODE HERE --------------------
covar = zeros(size(x, 1)); % You need to compute this
covar(:,:) = cov(xRot');

% Visualise the covariance matrix. You should see a line across the
% diagonal against a blue background.
figure('name','Visualisation of covariance matrix');
imagesc(covar);

%%=================================================================
```



Step 2: Find k, the number of components to retain

Write code to determine k, the number of components to retain in order to retain at least 99% of the variance.

```
% --------------------- YOUR CODE HERE --------------------
k = 144; % Set k accordingly
sqrt_k = sqrt(k);
variance_original = var(x,1);
variance_k = variance_original;
perc = min(variance_original ./ variance_k);
while sqrt_k > 0
    sqrt_k = sqrt_k - 1;
    xTilde = U(:,1:sqrt_k)' * x;
    variance_k = var(xTilde,1);
    perc = min(variance_original ./ variance_k);
    fprintf('Variance ratio for %d is %f\n', sqrt_k ^ 2, perc);
```

```
    if perc < .99
        k = (sqrt_k + 1) ^ 2;
        break;
    end
end

%%=================================================================
```

```
Variance ratio for 121 is 0.078062
```

Step 3: Implement PCA with dimension reduction

Now that you have found k, you can reduce the dimension of the data by discarding the remaining dimensions. In this way, you can represent the data in k dimensions instead of the original 144, which will save you computational time when running learning algorithms on the reduced representation.

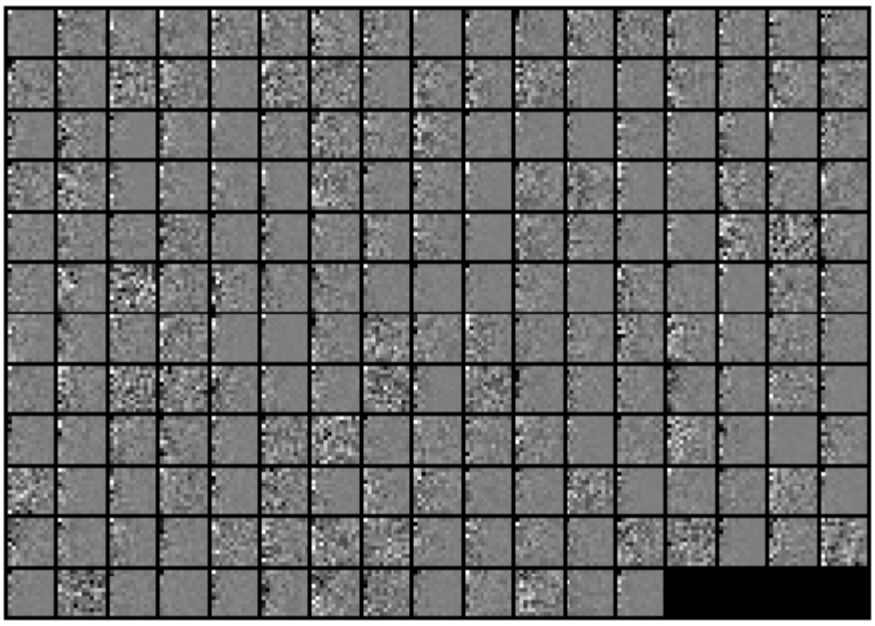
Following the dimension reduction, invert the PCA transformation to produce the matrix xHat, the dimension-reduced data with respect to the original basis. Visualise the data and compare it to the raw data. You will observe that there is little loss due to throwing away the principal components that correspond to dimensions with low variation.

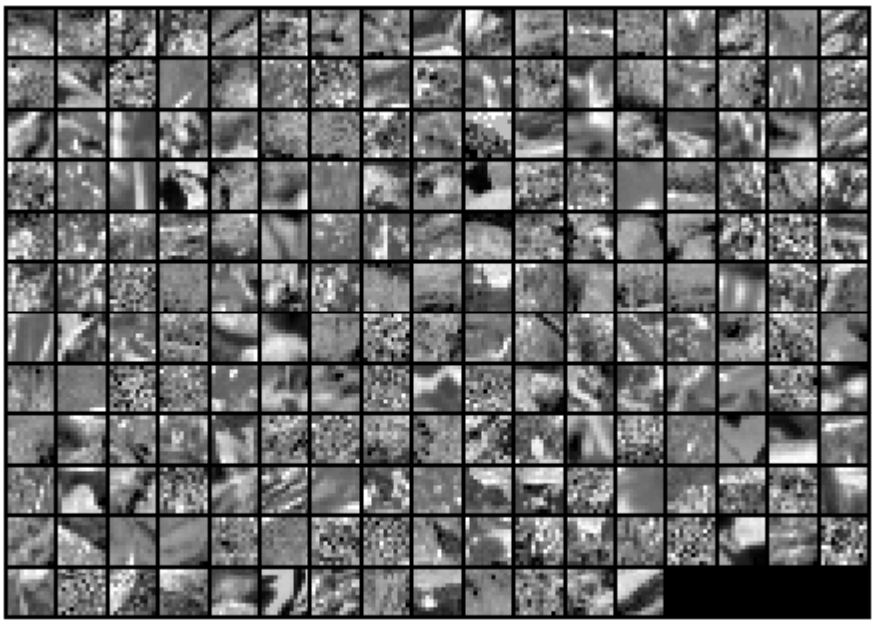
```
% -------------------- YOUR CODE HERE --------------------
xHat = U(:,1:k)' * x; % reduced dimension representation of the data,
                      % where k is the number of eigenvectors to keep

% Visualise the data, and compare it to the raw data
% You should observe that the raw and processed data are of comparable quality.
% For comparison, you may wish to generate a PCA reduced image which
% retains only 90% of the variance.

figure('name',['PCA processed images ',sprintf('(%d / %d dimensions)', k, size(x, 1)),'']);
display_network(xHat(:,randsel));
figure('name','Raw images');
display_network(x(:,randsel));

%%=================================================================
```





Step 4a: Implement PCA with whitening and regularisation

Implement PCA with whitening and regularisation to produce the matrix xPCAWhite.

```
epsilon = 0.1;
xPCAWhite = zeros(size(x));

% -------------------- YOUR CODE HERE --------------------
xPCAWhite(:,:) = diag(1./sqrt(diag(S) + epsilon)) * U' * x;

%%=================================================================
```

Step 4b: Check your implementation of PCA whitening

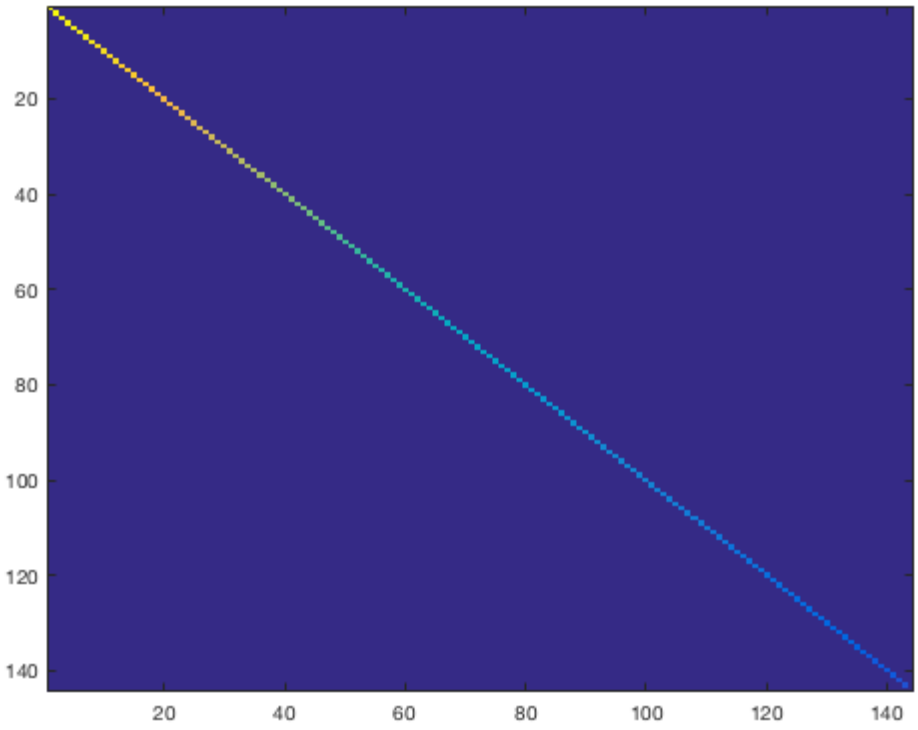
```
Check your implementation of PCA whitening with and without regularisation.
PCA whitening without regularisation results a covariance matrix
that is equal to the identity matrix. PCA whitening with regularisation
results in a covariance matrix with diagonal entries starting close to
1 and gradually becoming smaller. We will verify these properties here.
Write code to compute the covariance matrix, covar.

Without regularisation (set epsilon to 0 or close to 0),
when visualised as an image, you should see a red line across the
diagonal (one entries) against a blue background (zero entries).
With regularisation, you should see a red line that slowly turns
blue across the diagonal, corresponding to the one entries slowly
becoming smaller.
```

```
% -------------------- YOUR CODE HERE --------------------
covar(:,:) = cov(xPCAWhite');

% Visualise the covariance matrix. You should see a red line across the
% diagonal against a blue background.
figure('name','Visualisation of covariance matrix');
imagesc(covar);

%%=================================================================
```



Step 5: Implement ZCA whitening

Now implement ZCA whitening to produce the matrix xZCAWhite.
Visualise the data and compare it to the raw data. You should observe
that whitening results in, among other things, enhanced edges.

```
epsilon = 0.1;
xZCAWhite = zeros(size(x));
xZCAWhite = U * diag(1./sqrt(diag(S) + epsilon)) * U' * x;

% -------------------- YOUR CODE HERE --------------------

% Visualise the data, and compare it to the raw data.
% You should observe that the whitened images have enhanced edges.
figure('name','ZCA whitened images');
display_network(xZCAWhite(:,randsel));
figure('name','Raw images');
display_network(x(:,randsel));
```