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