Table of Contents


```
clc; clear;
```

Part 1.1

First, play around: (a) Load and run the MLP regression code. You have to use MATLAB with the Deep Learning Toolbox and running version 2019b or newer—MATLAB in the cloud will work, but interaction with the toolbox window might be slow. (b) Study the plot. What does “loss” seem to represent? Is it going down monotonically? Is there a difference between training and validation performance? Explain what you observe! (c) Change some of the optimization parameters: What happens if you pick an initial learning rate that is 10x, 100x bigger/smaller? Can you find out what an “Epoch” is? What does sgdm stand for? Replace sgdm with a different choice (e.g., adam with much bigger initial learning rate, e.g., 0.1), and see what happens. (d) Remove a layer of nodes / Add another layer of nodes / Change the number of nodes in each layer (e.g., 5, 20, or 50). What happens? (e) Repeat the above for the MLP classification code
****

(a) OK. (b) Loss seems to be just the RMSE (Root Mean Squared Error) or something related, ie. probably just MSE. It doesn't quite go down entirely monontonically but has some bumps sometimes. This is probably due to the momentum, or a bad batch. (c) Starting with lr = e-2 causes the training graph to have just one huge spike from overcorrecting at loss on order of e32 and then nothing, the table shows NaN, so possibly infinite error? Starting with small e-8 learning rate causes it to learn slowly. Epoch is # times through data, sgdm is stochastic gradient descent. Adam converges very quickly. And does a good job. (d) I added two new layers and validation rmse was about 0.03 which is actually worse than before, so it probably gets stuck in a local optima. The same thing happens when I set the number of nodes in the first layer to 50, it gets stuck at rmse of 0.17.

Part 1.2

First, play around: (a) Load and run the MLP regression code. You have to use MATLAB with the Deep Learning Toolbox and running version 2019b or newer—MATLAB in the cloud will work, but interaction with the toolbox window might be slow. (b) Study the plot. What does “loss” seem to represent? Is it going down monotonically? Is there a difference between training and validation performance? Explain what you observe! (c) Change some of the optimization parameters: What happens if you pick an initial learning rate that is 10x, 100x bigger/smaller? Can you find out what an “Epoch” is? What does sgdm stand for? Replace sgdm with a different choice (e.g., adam with much bigger initial learning rate, e.g., 0.1), and see what happens. (d) Remove a layer of nodes / Add another layer of nodes / Change the number of nodes in each layer (e.g., 5, 20, or 50). What happens? (e) Repeat the above for the MLP classification code
****

(a) OK. (b) The loss plot/accuracy is a lot more jumpy for classification. I am not sure why. This could be because they are computed discretely (you can't half correctly get the answer). But that was surprising. It kind of shark fins around a bit. (c) At e-4 learning rate, it essentially does not learn. Hovering at like 50% for a while. At e-9 practically nothing happens. It is just a flat curve. It does **slowly** go up in accuracy, and it is monotone (as far as I can see). But it progresses a lot slower. At 1000 epochs it is only at about 65% (d) I removed the 20 node layer. It actually seems to do pretty good, getting around 97% on validation.

Part 2

Build/train a two-input–two-output regression network for Cartesian to polar conversion: n(x, y) ≈ cart2pol(x, y). Plot the learned radius and angle landscape to "see" how good the training works. **

The network does a pretty poor job of learning the radius. It struggles around some lines for some reason (different each time). It kind of looks like a blob. The angle is not learned pretty well. At pi and -pi it has a hard time, because there is a discontinuity and around 0. But overall, it looks like a fairly smooth gradient across the space. This is surprising as it is a fairly non-trivial relation to learn (arcsin / arccos).

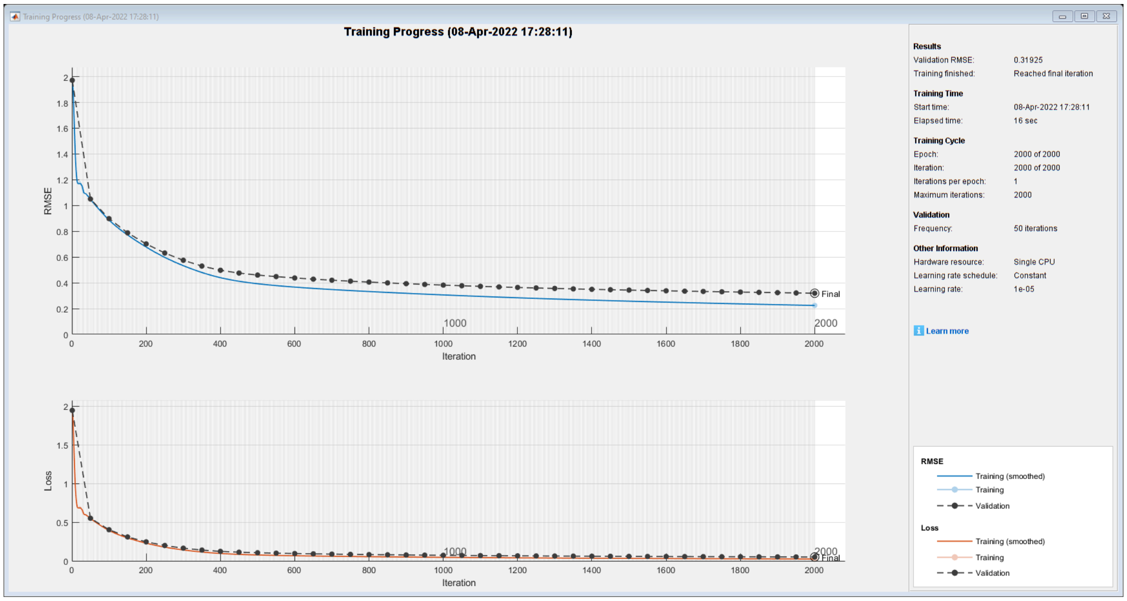
```
% setting up two hidden layers with 10 nodes, each, all fully
 connected
layers = [ sequenceInputLayer( 2 )
    fullyConnectedLayer(20)
    tanhLayer
    fullyConnectedLayer(10)
    tanhLayer
    fullyConnectedLayer(10)
    tanhLayer
    fullyConnectedLayer(2)
    regressionLayer
    ]

% n points in WxW square centered around 0
n = 1000;
W = 2;
x = rand(1,n) * W - W/2;
y = rand(1,n) * W - W/2;
XTrain = [x ; y];
[th, ro] = cart2pol(x, y);
YTrain = [th ; ro];

x = rand(1,n/10) * W - W/2;
y = rand(1,n/10) * W - W/2;
XVal = [x ; y];
[th, ro] = cart2pol(x, y);
YVal = [th ; ro];

% training options
options = trainingOptions('sgdm', ...
    'MaxEpochs',2000,...
    'InitialLearnRate',1e-5, ...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'ValidationData', {XVal,YVal} );
```

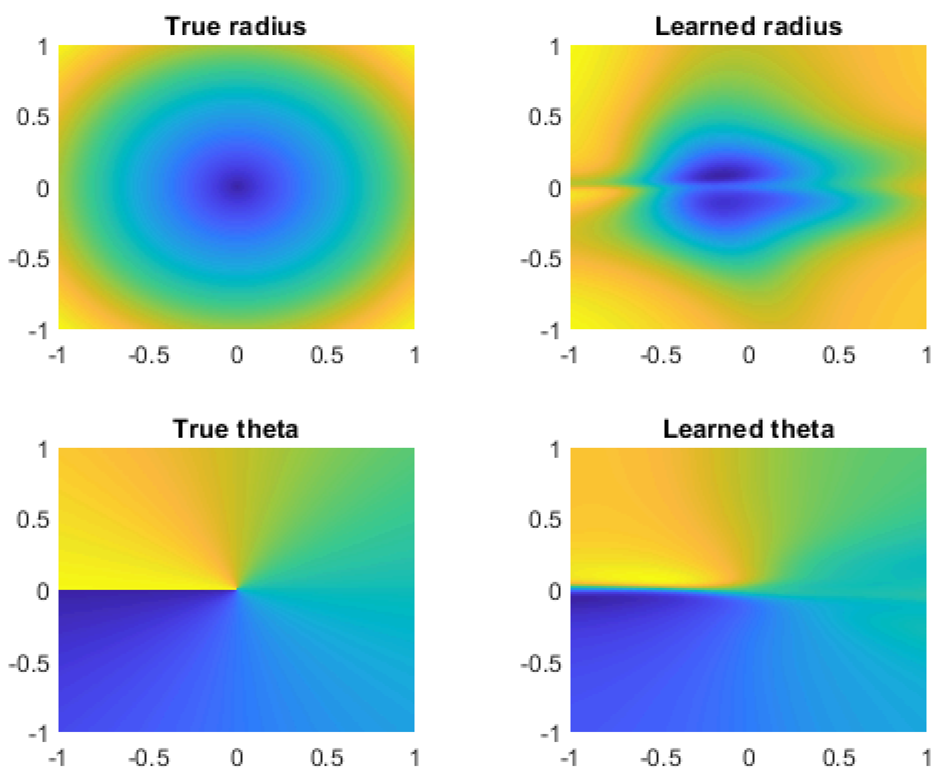
```
net = trainNetwork( XTrain, YTrain, layers, options );

plotX = -1:0.01:1;
plotY = -1:0.01:1;
[X, Y] = meshgrid(plotX, plotY);
XTest = [X(:)' ; Y(:)'];
[thv, rov] = cart2pol(X(:)' , Y(:)');

th = reshape(thv, size(X));
ro = reshape(rov, size(X));

preds = net.predict(XTest);
lthv = preds(1,:);
lrov = preds(2,:);

lth = reshape(lthv, size(X));
lro = reshape(lrov, size(X));

figure();
tiledlayout(2,2);

nexttile;
h = pcolor(plotX, plotY, ro);
set(h, 'EdgeColor', 'none');
title('True radius');

nexttile;
h = pcolor(plotX, plotY, lro);
set(h, 'EdgeColor', 'none');
title('Learned radius');

nexttile;
h = pcolor(plotX, plotY, th);
set(h, 'EdgeColor', 'none');
title('True theta');

nexttile;
h = pcolor(plotX, plotY, lth);
set(h, 'EdgeColor', 'none');
title('Learned theta');
```

```
layers =

  9x1 Layer array with layers:

     1   ''   Sequence Input      Sequence input with 2 dimensions
     2   ''   Fully Connected     20 fully connected layer
     3   ''   Tanh                Hyperbolic tangent
     4   ''   Fully Connected     10 fully connected layer
     5   ''   Tanh                Hyperbolic tangent
     6   ''   Fully Connected     10 fully connected layer
     7   ''   Tanh                Hyperbolic tangent
     8   ''   Fully Connected     2 fully connected layer
```

```
     9   ''   Regression Output   mean-squared-error
Training on single CPU.
|
===============================================================================
|  Epoch   |  Iteration   |  Time Elapsed   |  Mini-batch   |  Validation
 |  Mini-batch   |  Validation   |  Base Learning   |
|          |              |   (hh:mm:ss)    |     RMSE      |     RMSE
 |     Loss      |     Loss      |      Rate        |
|
===============================================================================
|       1 |            1 |        00:00:00 |         1.98 |         1.97
 |       1.9701 |       1.9441 |      1.0000e-05 |
|      50 |           50 |        00:00:00 |         1.04 |         1.05
 |       0.5458 |       0.5513 |      1.0000e-05 |
|     100 |          100 |        00:00:01 |         0.89 |         0.90
 |       0.3941 |       0.4034 |      1.0000e-05 |
|     150 |          150 |        00:00:01 |         0.77 |         0.79
 |       0.2978 |       0.3103 |      1.0000e-05 |
|     200 |          200 |        00:00:01 |         0.68 |         0.70
 |       0.2297 |       0.2465 |      1.0000e-05 |
|     250 |          250 |        00:00:02 |         0.60 |         0.63
 |       0.1787 |       0.1993 |      1.0000e-05 |
|     300 |          300 |        00:00:02 |         0.53 |         0.57
 |       0.1422 |       0.1653 |      1.0000e-05 |
|     350 |          350 |        00:00:03 |         0.48 |         0.53
 |       0.1154 |       0.1405 |      1.0000e-05 |
|     400 |          400 |        00:00:03 |         0.44 |         0.50
 |       0.0968 |       0.1236 |      1.0000e-05 |
|     450 |          450 |        00:00:04 |         0.41 |         0.48
 |       0.0849 |       0.1131 |      1.0000e-05 |
|     500 |          500 |        00:00:04 |         0.39 |         0.46
 |       0.0772 |       0.1059 |      1.0000e-05 |
|     550 |          550 |        00:00:04 |         0.38 |         0.45
 |       0.0717 |       0.1004 |      1.0000e-05 |
|     600 |          600 |        00:00:05 |         0.37 |         0.44
 |       0.0673 |       0.0958 |      1.0000e-05 |
|     650 |          650 |        00:00:05 |         0.36 |         0.43
 |       0.0637 |       0.0918 |      1.0000e-05 |
|     700 |          700 |        00:00:06 |         0.35 |         0.42
 |       0.0606 |       0.0883 |      1.0000e-05 |
|     750 |          750 |        00:00:06 |         0.34 |         0.41
 |       0.0578 |       0.0852 |      1.0000e-05 |
|     800 |          800 |        00:00:07 |         0.33 |         0.41
 |       0.0553 |       0.0823 |      1.0000e-05 |
|     850 |          850 |        00:00:07 |         0.33 |         0.40
 |       0.0530 |       0.0797 |      1.0000e-05 |
|     900 |          900 |        00:00:08 |         0.32 |         0.39
 |       0.0508 |       0.0774 |      1.0000e-05 |
|     950 |          950 |        00:00:08 |         0.31 |         0.39
 |       0.0488 |       0.0752 |      1.0000e-05 |
|    1000 |         1000 |        00:00:09 |         0.31 |         0.38
 |       0.0469 |       0.0731 |      1.0000e-05 |
|    1050 |         1050 |        00:00:09 |         0.30 |         0.38
 |       0.0451 |       0.0712 |      1.0000e-05 |
```

1100	1100	00:00:09	0.29	0.37	0.0434	0.0694	1.0000e-05
1150	1150	00:00:10	0.29	0.37	0.0418	0.0678	1.0000e-05
1200	1200	00:00:10	0.28	0.36	0.0403	0.0663	1.0000e-05
1250	1250	00:00:11	0.28	0.36	0.0389	0.0648	1.0000e-05
1300	1300	00:00:11	0.27	0.36	0.0376	0.0635	1.0000e-05
1350	1350	00:00:11	0.27	0.35	0.0364	0.0623	1.0000e-05
1400	1400	00:00:12	0.27	0.35	0.0353	0.0611	1.0000e-05
1450	1450	00:00:12	0.26	0.35	0.0342	0.0600	1.0000e-05
1500	1500	00:00:13	0.26	0.34	0.0332	0.0590	1.0000e-05
1550	1550	00:00:13	0.25	0.34	0.0322	0.0581	1.0000e-05
1600	1600	00:00:13	0.25	0.34	0.0313	0.0571	1.0000e-05
1650	1650	00:00:14	0.25	0.34	0.0304	0.0563	1.0000e-05
1700	1700	00:00:14	0.24	0.33	0.0296	0.0554	1.0000e-05
1750	1750	00:00:14	0.24	0.33	0.0288	0.0546	1.0000e-05
1800	1800	00:00:15	0.24	0.33	0.0280	0.0539	1.0000e-05
1850	1850	00:00:15	0.23	0.33	0.0272	0.0531	1.0000e-05
1900	1900	00:00:16	0.23	0.32	0.0265	0.0524	1.0000e-05
1950	1950	00:00:16	0.23	0.32	0.0258	0.0517	1.0000e-05
2000	2000	00:00:16	0.22	0.32	0.0251	0.0510	1.0000e-05





Part 3

Pick CBCL1 or news data, and build a binary classifier. Put randomly selected portions of the data (10%, each) away for validation and testing (network trains on 80% data, validates on 10% while training, and when done, you run the network on the 10% left out for testing). What percent correct can you achieve on the training/validation/testing data? How does this com-

pare against SVM (train/test SVM on the same data partitions used for the network). Page 4
****

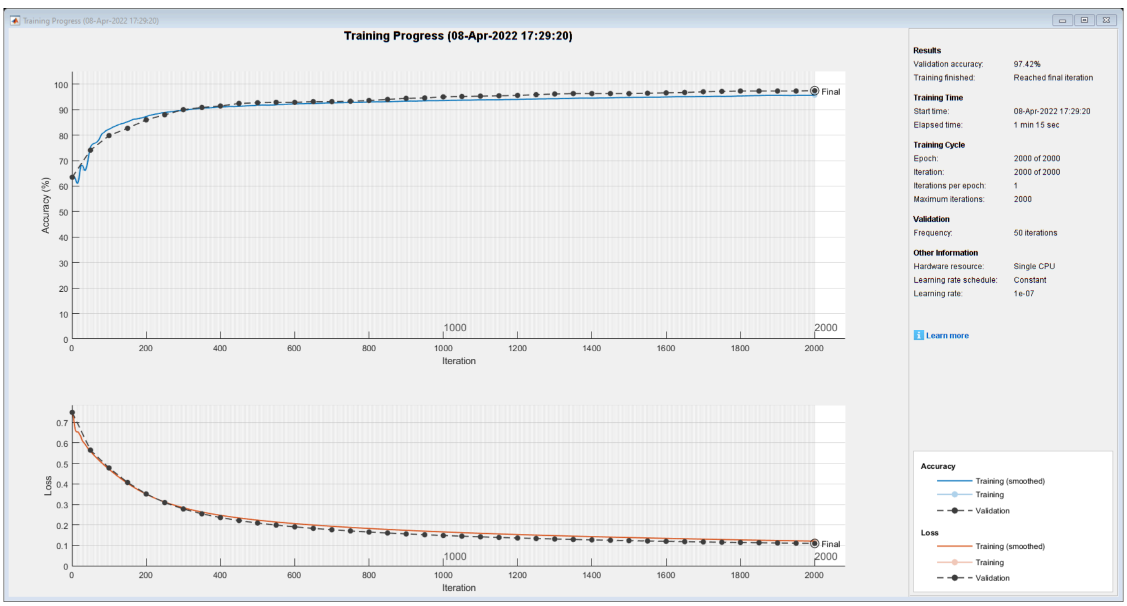
Initially, the neural net performed pretty bad. I think it was too small. So I increased the number of nodes in the first layer and also increased the depth of the network. Now it performs pretty well. Not that much better than SVM so it is maybe not worth the complexity.

```
clear;
load('cbcl1.mat')

ii = randperm(size(X,2));
trainIdx = ii(1: floor(0.80 * length(ii)));
valIdx = ii(floor(0.80 * length(ii)) + 1 : floor(0.90 * length(ii)));
testIdx = ii(floor(0.90 * length(ii))+1 : end);

XTrain = X(:, trainIdx);
YTrain = L(trainIdx)';
XVal = X(:, valIdx);
YVal = L(valIdx)';
XTest = X(:, testIdx);
YTest = L(testIdx)';

[w, b, xi] = softsvm(XTrain,YTrain', 0.005);
SVMpreds = (XTest' * w + b) > 0;

layers = [ sequenceInputLayer( size(X,1) )
    fullyConnectedLayer(100)
    tanhLayer
    fullyConnectedLayer(70)
    tanhLayer
    fullyConnectedLayer(50)
    tanhLayer
    fullyConnectedLayer(2)  % there are two classes, so two of these
 nodes
    softmaxLayer    %
    classificationLayer   % these two are needed for classification
 output
 ]


% training options
options = trainingOptions('sgdm', ...
    'MaxEpochs',2000,...
    'InitialLearnRate',1e-7, ...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'ValidationData', {XVal, categorical(YVal)} );

net = trainNetwork( XTrain, categorical(YTrain), layers, options );
NETpreds = net.classify(XTest);

netAccuracy = sum(NETpreds == categorical(YTest)) / length(YTest)
svmAccuracy = sum(SVMpreds' == (YTest > 0)) / length(YTest)
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint
 tolerance.


layers =

  10x1 Layer array with layers:

     1   ''   Sequence Input          Sequence input with 361
 dimensions
     2   ''   Fully Connected         100 fully connected layer
     3   ''   Tanh                    Hyperbolic tangent
     4   ''   Fully Connected         70 fully connected layer
     5   ''   Tanh                    Hyperbolic tangent
     6   ''   Fully Connected         50 fully connected layer
     7   ''   Tanh                    Hyperbolic tangent
     8   ''   Fully Connected         2 fully connected layer
     9   ''   Softmax                 softmax
    10   ''   Classification Output   crossentropyex
Training on single CPU.
|
=================================================================================
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation
 |  Mini-batch  |  Validation  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |   Accuracy
 |     Loss     |     Loss     |      Rate       |
|
=================================================================================
|       1 |           1 |       00:00:00 |       64.27% |       63.47%
 |       0.7509 |       0.7492 |      1.0000e-07 |
|      50 |          50 |       00:00:02 |       75.24% |       74.07%
 |       0.5590 |       0.5650 |      1.0000e-07 |
|     100 |         100 |       00:00:04 |       82.28% |       79.80%
 |       0.4716 |       0.4781 |      1.0000e-07 |
|     150 |         150 |       00:00:06 |       85.22% |       82.66%
 |       0.4030 |       0.4073 |      1.0000e-07 |
|     200 |         200 |       00:00:07 |       87.30% |       85.96%
 |       0.3502 |       0.3511 |      1.0000e-07 |
|     250 |         250 |       00:00:09 |       88.87% |       87.97%
 |       0.3118 |       0.3091 |      1.0000e-07 |
|     300 |         300 |       00:00:11 |       89.70% |       89.97%
 |       0.2839 |       0.2778 |      1.0000e-07 |
|     350 |         350 |       00:00:13 |       90.52% |       90.83%
 |       0.2630 |       0.2541 |      1.0000e-07 |
|     400 |         400 |       00:00:15 |       91.02% |       91.40%
 |       0.2468 |       0.2358 |      1.0000e-07 |
|     450 |         450 |       00:00:17 |       91.29% |       92.41%
 |       0.2339 |       0.2211 |      1.0000e-07 |
```

500	500	00:00:19	91.74%	92.69%	0.2232	0.2091	1.0000e-07
550	550	00:00:21	91.92%	92.84%	0.2142	0.1990	1.0000e-07
600	600	00:00:22	92.21%	92.84%	0.2064	0.1905	1.0000e-07
650	650	00:00:24	92.38%	93.12%	0.1994	0.1830	1.0000e-07
700	700	00:00:26	92.64%	93.12%	0.1933	0.1764	1.0000e-07
750	750	00:00:28	92.81%	93.27%	0.1877	0.1705	1.0000e-07
800	800	00:00:30	92.98%	93.55%	0.1826	0.1652	1.0000e-07
850	850	00:00:32	93.07%	93.98%	0.1779	0.1604	1.0000e-07
900	900	00:00:34	93.32%	94.41%	0.1736	0.1560	1.0000e-07
950	950	00:00:36	93.44%	94.56%	0.1696	0.1520	1.0000e-07
1000	1000	00:00:37	93.53%	94.99%	0.1658	0.1483	1.0000e-07
1050	1050	00:00:39	93.71%	95.13%	0.1623	0.1449	1.0000e-07
1100	1100	00:00:41	93.80%	95.27%	0.1590	0.1418	1.0000e-07
1150	1150	00:00:43	93.85%	95.42%	0.1559	0.1389	1.0000e-07
1200	1200	00:00:45	93.98%	95.56%	0.1530	0.1361	1.0000e-07
1250	1250	00:00:47	94.14%	95.85%	0.1502	0.1336	1.0000e-07
1300	1300	00:00:49	94.27%	96.13%	0.1476	0.1312	1.0000e-07
1350	1350	00:00:51	94.43%	96.28%	0.1451	0.1290	1.0000e-07
1400	1400	00:00:52	94.52%	96.28%	0.1427	0.1269	1.0000e-07
1450	1450	00:00:54	94.62%	96.28%	0.1405	0.1250	1.0000e-07
1500	1500	00:00:56	94.75%	96.28%	0.1383	0.1232	1.0000e-07
1550	1550	00:00:58	94.84%	96.42%	0.1362	0.1214	1.0000e-07
1600	1600	00:01:00	94.97%	96.56%	0.1343	0.1198	1.0000e-07
1650	1650	00:01:02	95.05%	96.70%	0.1324	0.1183	1.0000e-07
1700	1700	00:01:04	95.11%	96.99%	0.1305	0.1168	1.0000e-07
1750	1750	00:01:06	95.18%	97.13%	0.1288	0.1155	1.0000e-07
1800	1800	00:01:07	95.32%	97.28%	0.1271	0.1142	1.0000e-07

```
|    1850 |        1850 |       00:01:09 |       95.50% |       97.28%
 |       0.1254 |       0.1129 |      1.0000e-07 |
|    1900 |        1900 |       00:01:11 |       95.59% |       97.28%
 |       0.1239 |       0.1118 |      1.0000e-07 |
|    1950 |        1950 |       00:01:13 |       95.56% |       97.28%
 |       0.1223 |       0.1106 |      1.0000e-07 |
|    2000 |        2000 |       00:01:15 |       95.66% |       97.42%
 |       0.1209 |       0.1096 |      1.0000e-07 |
|
=================================================================================

netAccuracy =

    0.9613


svmAccuracy =

    0.9355
```



softsvm.m

```
%SOFTSVM    Learns an approximately separating hyperplane for the
 provided data.
% [w, b, xi] = softsvm( X, l, gamma )
%
% Input:
% X : D x N matrix of data points
% l : N x 1 vector with class labels (+/- 1)
% gamma : scalar slack variable penalty
%
```

```
% Output:
% w : D x 1 vector normal to the separating hyperplane
% b : scalar offset
% xi : N x 1 vector of slack variables
%
% classify data using sign( X'*w + b )

function [w, b, xi] = softsvm( X, l, gamma )

[D,N] = size(X);

% construct H, f, A, b, and lb


gamma = 0.005;
% Quadratic Objective
H = spdiags([zeros(N,1); ones(D,1); 0] , 0, N + D + 1, N + D + 1);
% Linear Objective
f =[gamma * ones(N, 1); zeros(D,1); 0];  % gamma N times, then D+1
 zeros for right shape
% Linear Innequality Constraints Ax <= b
L = spdiags(l, 0, N, N );
A = -1 * [speye(N), L*X', l];
b = -1 * ones(N, 1);
% Lower bounds
lb = [zeros(N, 1); -Inf(D+1,1)];

% Solve
x = quadprog( H, f, A, b, [], [], lb );

% distribute components of x into w, b, and xi:
xi = x(1:N);
w = x(N+1:N+D);
b = x(end);
end
```

MLP Classification.m

```
sigma = 0.45;

[XTrain, YTrain] = swissroll(1e5,sigma);
[XV, YV] = swissroll(1e4,sigma);

plotroll(XTrain, YTrain); % show the training data

% setting up two hidden layers with 10 nodes, each, all fully
 connected
layers = [ sequenceInputLayer( 2 ) % 2-component input
    fullyConnectedLayer(50)
    tanhLayer
    fullyConnectedLayer(30)
```

```
    tanhLayer
    fullyConnectedLayer(20)
    tanhLayer
    fullyConnectedLayer(2)  % there are two classes, so two of these
 nodes
    softmaxLayer    %
    classificationLayer   % these two are needed for classification
 output
    ]

% training options
options = trainingOptions('sgdm', ...
    'MaxEpochs',5000,...
    'InitialLearnRate',1e-7, ...
    'Momentum', 0.95,...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'ValidationData', {XV,categorical(YV)} );

% let MATLAB do the actual training
net = trainNetwork( XTrain, categorical(YTrain), layers, options );

% now use the trained network to paint entire feature space
[x1,x2]=meshgrid(linspace(-15,15,1000)); % testing x
XTest = [x1(:)'; x2(:)'];
y = net.classify(XTest); % network's prediction

YTest = zeros(1,size(XTest,2));  % convert categorical to numerical
 output
YTest(y == '1') = 1;
YTest(y == '-1') = -1;

figure; plotroll(XTest,YTest); % plot the classificatoin landscape
```

MLP Regression.m

```
% setting up two hidden layers with 10 nodes, each, all fully
 connected
layers = [ sequenceInputLayer( 1 )
    fullyConnectedLayer(10)
    tanhLayer
    fullyConnectedLayer(10)
    tanhLayer
    fullyConnectedLayer(1)
    regressionLayer
    ]

% training data
XTrain = 2*pi*rand(1,1e4); %  x, sampled uniformly from [0,2pi]
YTrain = sin(XTrain)+0.1*randn(size(XTrain)); % corresponding y w/
 noise
```

```
% validation data
XV = 2*pi*rand(1,1e3);
YV = sin(XV);

% training options
options = trainingOptions('sgdm', ...
    'MaxEpochs',2000,...
    'InitialLearnRate',1e-5, ...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'ValidationData', {XV,YV} );

% let MATLAB do the actual training
net = trainNetwork( XTrain, YTrain, layers, options );

% now use the trained network
x = 0:0.001:(2*pi); % testing x
y = net.predict(x); % network's prediction

figure;
plot( x, y ); hold on; % learned function
plot( x, sin(x) );     % true function
plot( XTrain(1:100:end), YTrain(1:100:end), 'ko'); % some data points
```

swissroll.m

```
function [X,Y] = swissroll(N,sigma)
    t = 2*randn(1,N)+7.5;
    Y = sign(randn(1,N));
    X = [t.*cos(t+pi/2*Y); t.*sin(t+pi/2*Y) ] + sigma*randn(2,N);
end
```

plotroll.m

```
function plotroll(X,Y)
    scatter(X(1,Y>0), X(2,Y>0),'bo'); hold on;
    scatter(X(1,Y<0), X(2,Y<0),'rx'); hold on;
    set(gca,'XLim', [-15,15],'YLim', [-15,15] );
    daspect([1 1 1]);
end
```

Published with MATLAB® R2020a