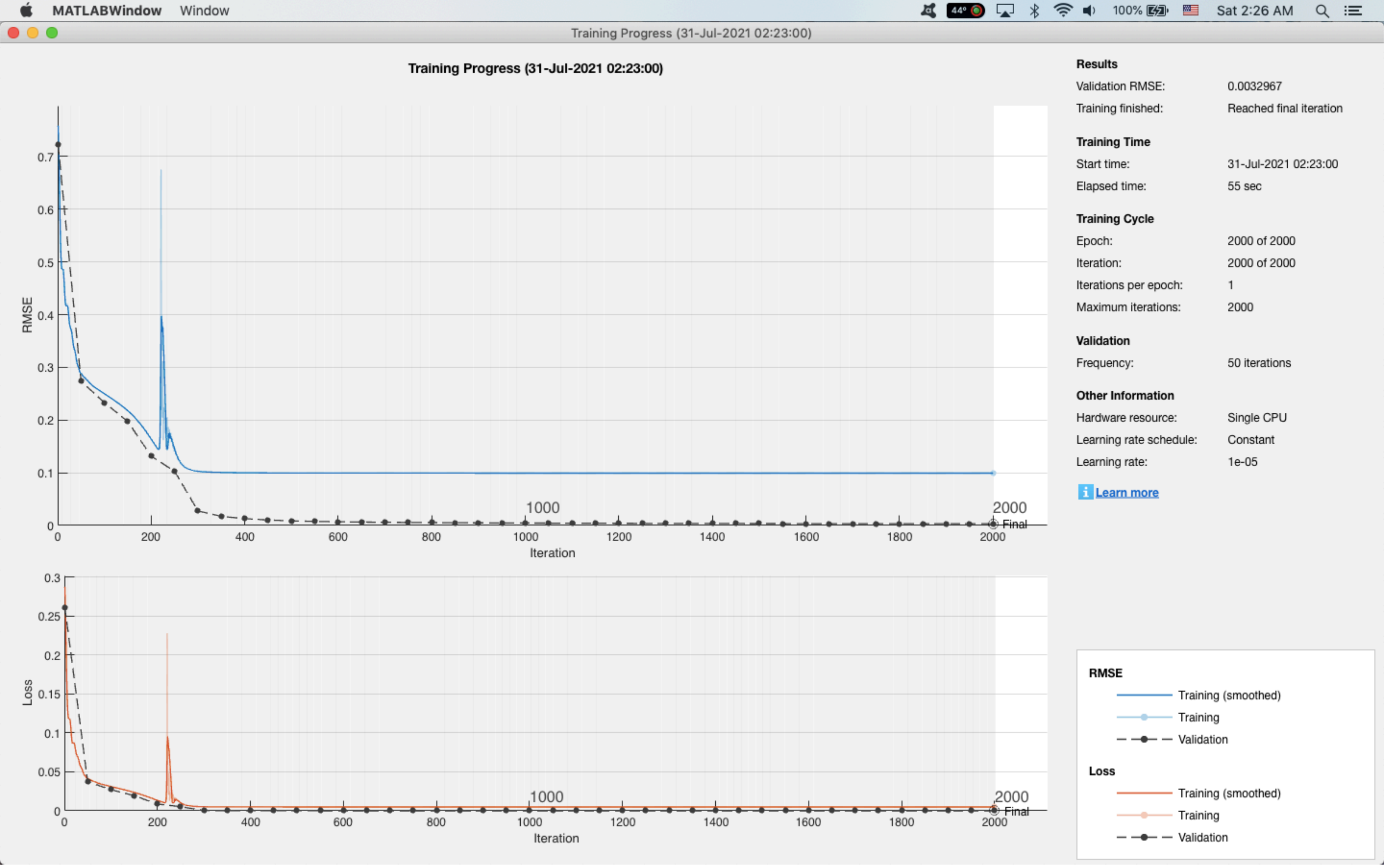
Contents


```
% setting up two hidden layers with 10 nodes, each, all fully connected
layers = [ sequenceInputLayer( 1 )
    fullyConnectedLayer(20)
    tanhLayer
    fullyConnectedLayer(20)
    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('adam', ...
    'MaxEpochs',2000,...
    'InitialLearnRate',0.1, ...
    '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
```

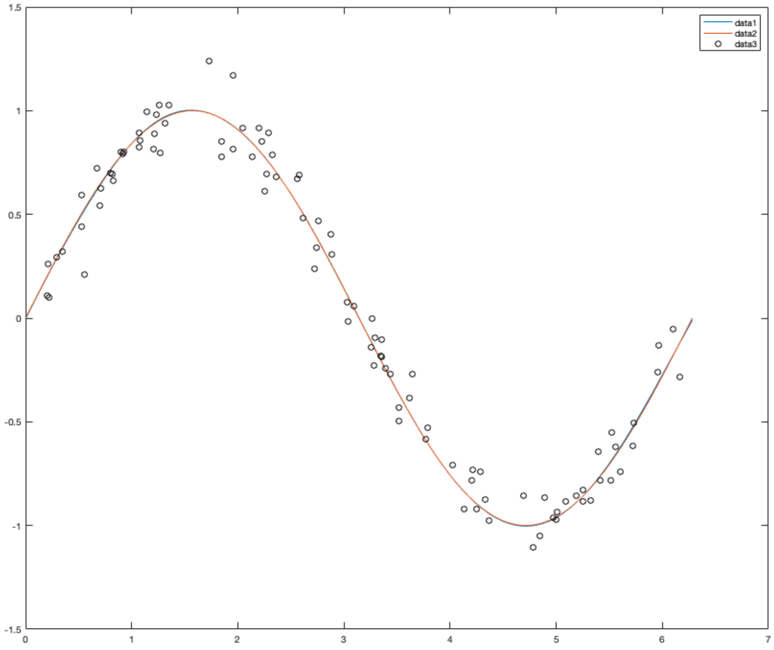
Observations:

```
%1.b) The observation we note just from the graphs of loss and root mean
%square error (RMSE) of running the NN is that loss decreases at similar
%rate as RMSE. They are mostly monotonically decreasing with occasional
%spikes, and the loss/RMSE is smaller for validation data than training.
%This is not too surprising however, since we generated the training data
%with noise and validation data without.

%The documentation writes that RMSE plot represent the RMSE of NN's
%prediction of the current batch ( = entire data set in this case) in each
%iteration of NN's training (and RMSE of validation set is computed once
%every 50 iterations). This is done while NN is training!

%Note: a batch may not be entire dataset, and it depends on user defined
%parameter: "iteration per epoch" = #datapts/BatchSize
```

```
%Here "iteration per epoch" = 1, so a batch is the entire data set.

%The loss is defined differently for various purposes of the NN, and in
%particular in regression here it is the MSE of NN's predictions,which is
%just the square of the RMSE. We see this relation in the plot: when RMSE
%is .1, MSE appears to be .01.

%1.c) Since the code to run will be repeat of before where we only replace
%some training parameters, I ran beforehand and what I observed:

%10x bigger learn rate: for some reason the learn rate is too big that
%after only a tens of iterations the neural net diverged...

%10x smaller learn rate: as expected, with lower learning rate we converge
%much slower to optimal solution, and after 2000 iteration both training
%and validation RMSE is still not quite minimized

%Epoch refers to a full pass of the data set to NN, and here since we set
%"iterations per epoch" = 1, i.e. have the batch in each iteration be the
%the full dataset, a full pass to data takes only 1 iteration.

%'sgdm' here stands for Stochastic Gradient Descent with Momentum. Momentum
%is like also considering a weighted amount of previous step for current
%step in descent. It provably often makes the learning occur quicker than
%without it since intuitively its speeds up the descent at steep directions
%by also adding the previous descent value.

%'adam' stands for ADAptive Momentum estimation, where the learning rates
%are adaptive to the mean and variance of gradient.

%Using 'adam' with learn rate 0.1: This scheme initially descends the loss
%function very quickly, but after 200 iterations it fails to manage the
%final stretch of minimization as successfully as the regular sgdm with
%1e-5 learn rate. There is also more fluctuations.

%1.d) I repeated the training for 5, 50 neurons per hidden layer, and for
%removing and adding hidden layers. Below are the observation:

%half neurons per layer: loss drops in similar pace with original, but
%the validation error still have some room to minimize.

%double neuron per layer: The performance of this is very similar to the
%original after 2000 iterations, but the loss dropped much quickly compared
%to the original in the initial iterations.

%1 layer removed: loss drops in similar pace with original initially, but
%it often have many spikes after the initial drop in loss.

%1 layer added: loss drops very quickly, but it also doesn't quite converge
%as well as original sgdm to minimum
```



Red - Actual sine
Blue - Sine from neural net
As shown, almost overlap!



Contents

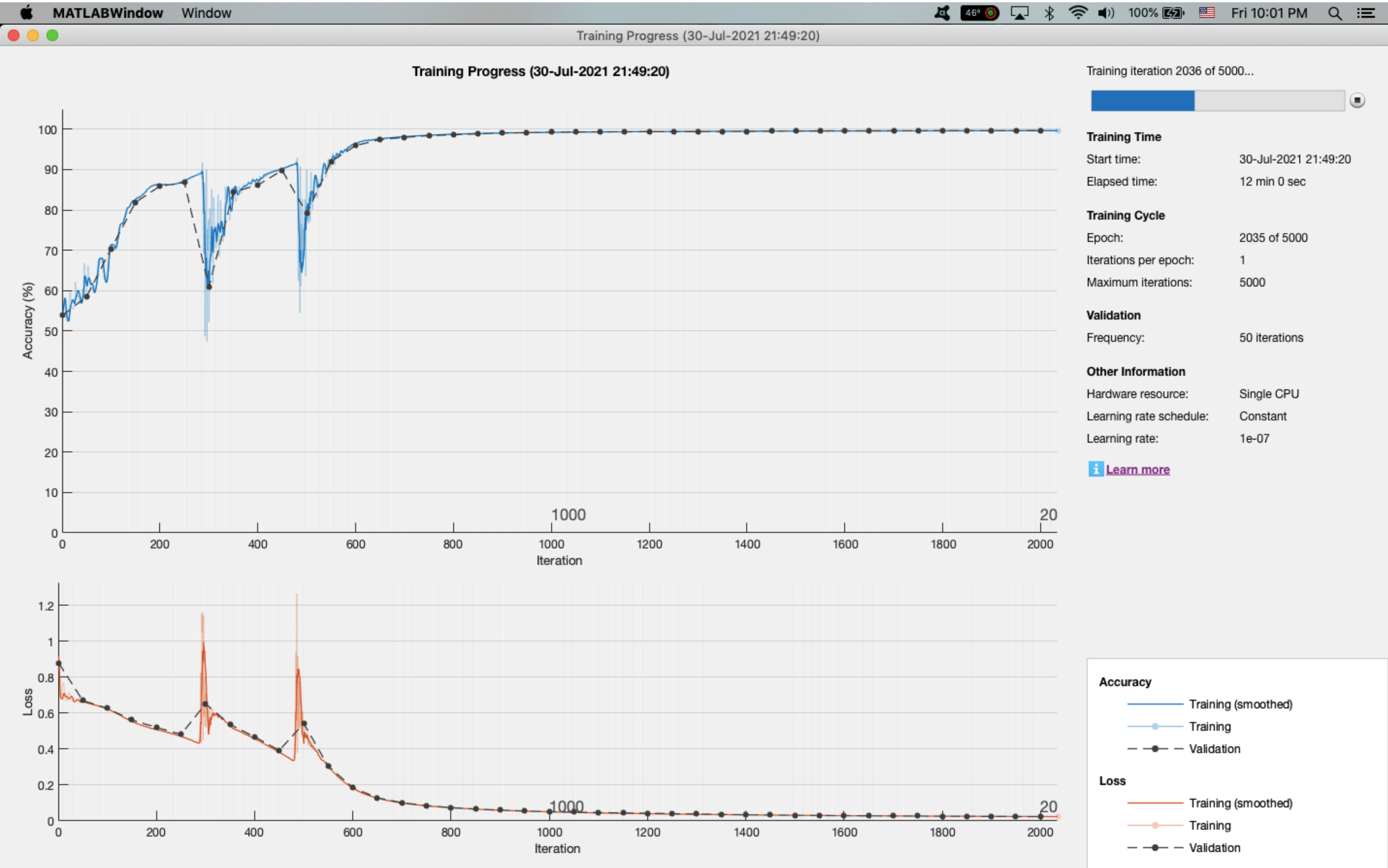

```
sigma = 0.45;

[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
```

1.5 Repeat 1.2-1.4 for classifying 2 types of swiss rolls patterns

```
%1.b) The observation we note just from the plot is that first, in this
%case accuracy is used as metric (instead of RMSE) as it is more meaningful
%in context of classification. We observe increasing accuracy corresponds
%to similar reduction in loss. Loss is not falling monotonically but still
%diminishing to near 0 in the limit. Here, NN performed similarly well on
%validation data as training data, and this is expected because of noise in
%training data, which is not in validation data.

%Here, training acc is percentage of correct classified points by NN out
```

```
%of the batch of entire dataset, in each iteration, (and validation set
%loss is computed every 50 iterations). From documentation, here we use
%cross entropy loss, more suitable for classification.

%1.c) Since the code to run will be repeat of before where we only replace
%some training parameters, I ran beforehand and what I observed:

%10x bigger learn rate: Similar explanation to regression

%10x smaller learn rate: Similar explanation to regression

%Using 'adam' with learn rate 0.1: Similar explanation to regression

%Epoch here is same as in MLP_regression.

%1.d) I repeated the training for 5, 50 neurons per hidden layer, and for
%removing and adding hidden layers. Below are the observation:

%half neurons per layer: Similar explanation to regression

%double neurons per layer: Similar explanation to regression

%1 layer removed: Similar explanation to regression

%1 layer added: Similar explanation to regression
```

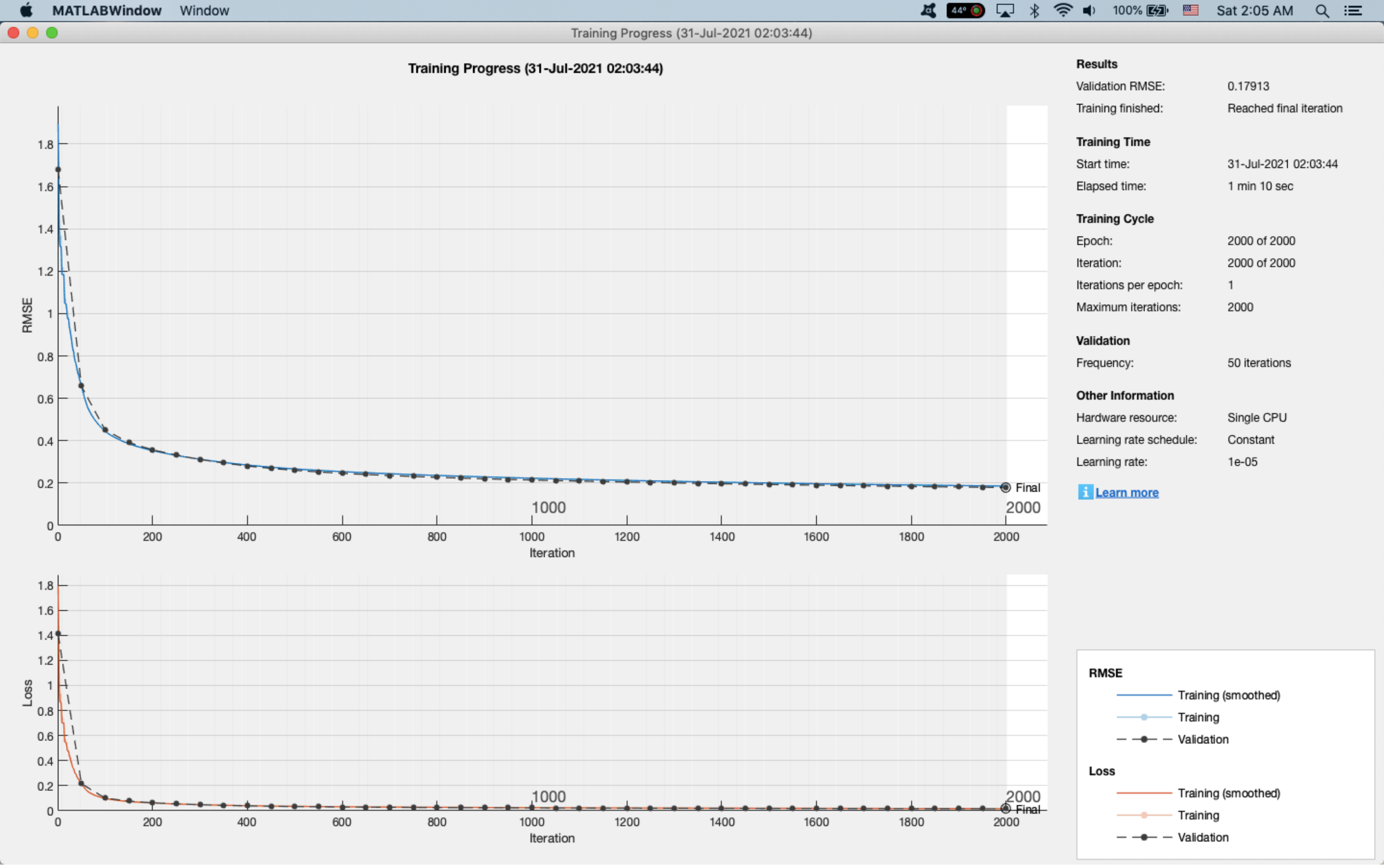
Published with MATLAB® R2020b


MATLABWindow
Window
Fri 10:01 PM
100%
Training Progress (30-Jul-2021 21:49:20)
Training Progress (30-Jul-2021 21:49:20)
Accuracy (%)
Iteration
Loss
Iteration
Training iteration 2036 of 5000...
Training Time
Start time: 30-Jul-2021 21:49:20
Elapsed time: 12 min 0 sec
Training Cycle
Epoch: 2035 of 5000
Iterations per epoch: 1
Maximum iterations: 5000
Validation
Frequency: 50 iterations
Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 1e-07
Learn more
Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation

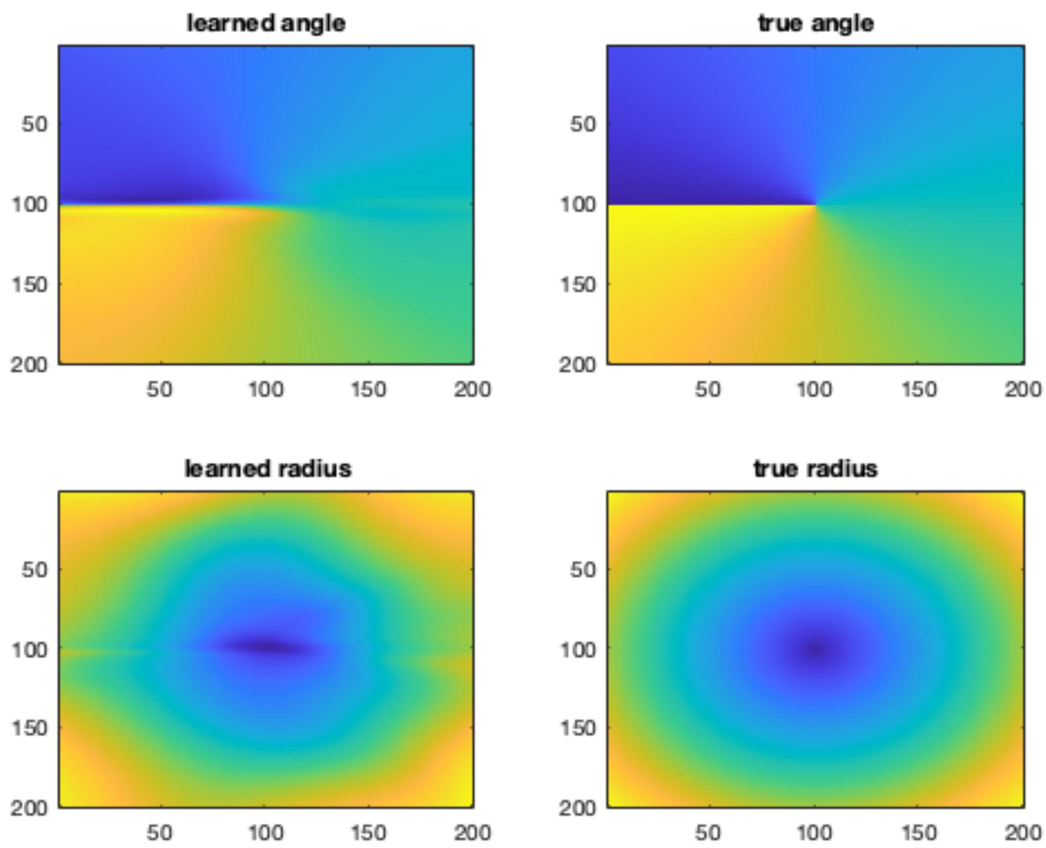
Contents

3.2 Build 2-input 2-output regression NN for cart2pol

```
% building NN same as in MLP_regression, but input/output is 2 nodes as
% (x,y) and (theta,r) are 2-D data.
layers = [ sequenceInputLayer( 2 )
    fullyConnectedLayer(10)
    tanhLayer
    fullyConnectedLayer(10)
    tanhLayer
    fullyConnectedLayer(2)
    regressionLayer
    ]

%Input: (x,y) in [-1,1]x[-1,1], Target: (theta,r) given (x,y)
%Output: the weights to infer (theta,r)

%Training Data:
XTrain = -1+2*rand(2,1e4); %points uniformly on [-1,1]x[-1,1]
[tTrain,rTrain] = cart2pol(XTrain(1,:), XTrain(2,:));
YTrain = [tTrain;rTrain];

%Validation Data:
XV = -1+2*rand(2,1e3);
[tV,rV] = cart2pol(XV(1,:), XV(2,:));
YV = [tV;rV];

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

% train network
net = trainNetwork(XTrain, YTrain, layers, options);

%generate grid of points in [-1,1]x[-1,1] with spacing 0.01.
[gridX, gridY] = meshgrid(-1:0.01:1);
X = [gridX(:)'; gridY(:)'];

%compute the actual cart2pol result of grid data
[THETA, R] = cart2pol(gridX,gridY);

%use trained network to learn cart2pol of grid data
Y = net.predict(X);

%Visualize radius and angle landscape of learned vs actual polar coords of
%the grid data generated
figure;
subplot(221);
imagesc(reshape(Y(1,:), size(gridX)));
```



MATLABWindow Window
Sat 2:05 AM
Training Progress (31-Jul-2021 02:03:44)
Training Progress (31-Jul-2021 02:03:44)
RMSE
Iteration
Loss
Iteration
Final
Results
Validation RMSE: 0.17913
Training finished: Reached final iteration
Training Time
Start time: 31-Jul-2021 02:03:44
Elapsed time: 1 min 10 sec
Training Cycle
Epoch: 2000 of 2000
Iteration: 2000 of 2000
Iterations per epoch: 1
Maximum iterations: 2000
Validation
Frequency: 50 iterations
Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 1e-05
Learn more
RMSE
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation



learned angle
true angle
learned radius
true radius

```
title('learned angle');
subplot(222);
imagesc(THETA);
title('true angle');
subplot(223);
imagesc(reshape(Y(2,:), size(gridX)));
title('learned radius');
subplot(224);
imagesc(R);
title('true radius');

%we note the training is pretty good since the learned radius and angle has
%landscape very similar to the actual landscape. Though from NN training
%plot we notice loss is still decreasing slightly at 2000 -> potential
%value in learning for more epochs.
```

3.3.1 Build D-input binary classifier NN for CBCL data

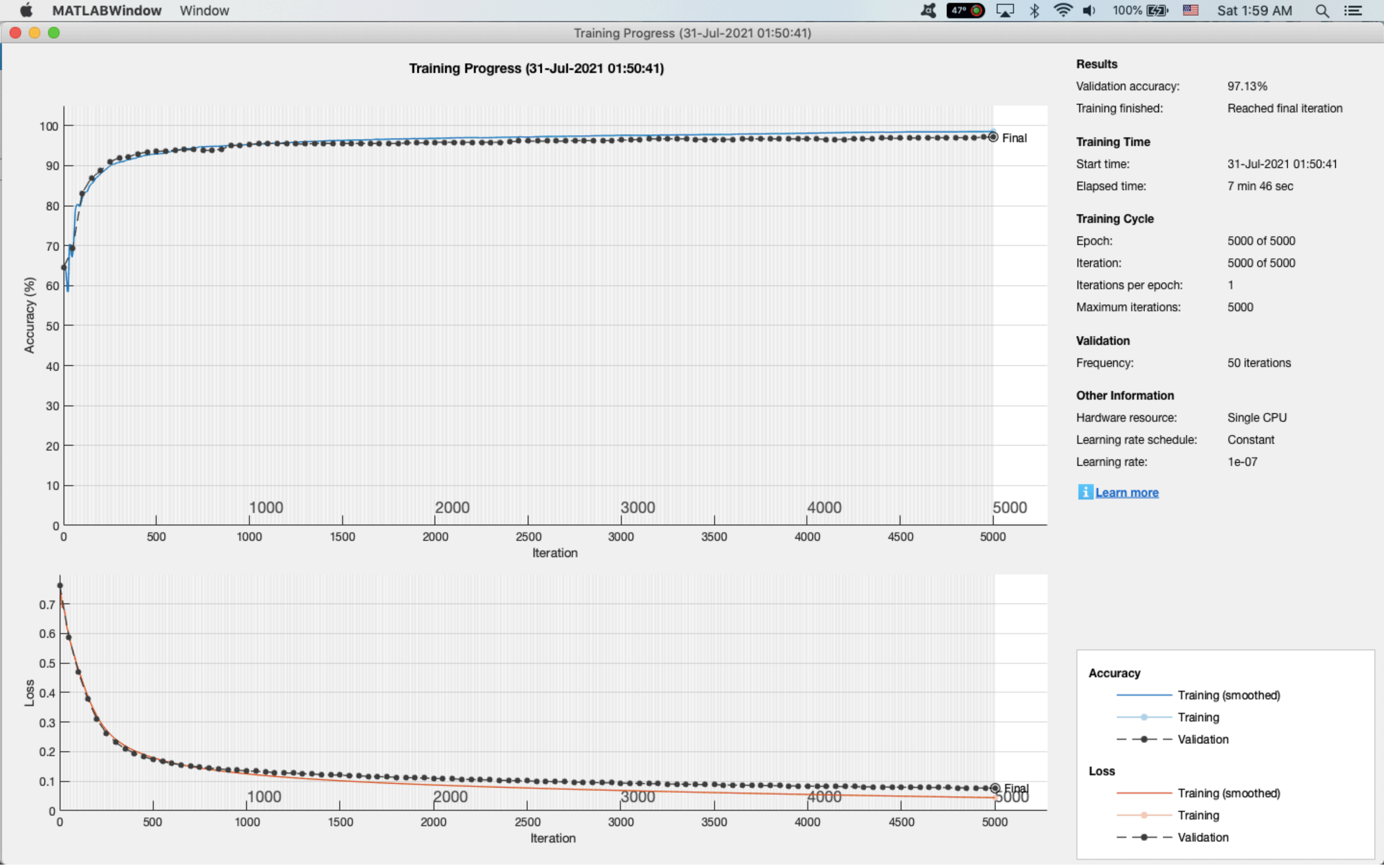
```
load 'cbcl.mat'

%D - dimension of data point, N - size of dataset
[D, N] = size(X);

%Shuffle and distribute data into train,validation,test
idx = randperm(N,N);
i80 = round(0.8*N);
i90 = round(0.9*N);

XTrain = X(:,idx(1:i80));
YTrain = L(idx(1:i80));

XV = X(:,idx(i80+1:i90));
YV = L(idx(i80+1:i90));

XTest = X(:,idx(i90+1:end));
YTest = L(idx(i90+1:end));

% building NN same as in MLP_classification, but input layer has D nodes
% since data is D-dimensional
layers = [ sequenceInputLayer( D ) % D-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, same as in MLP_classification
options = trainingOptions('sgdm', ...
    'MaxEpochs',5000,...
    'InitialLearnRate',1e-7, ...
    'Momentum', 0.95,...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'ValidationData', {XV,categorical(YV')} );
```

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

y = net.classify(XTest);

% convert categorical to numerical
yNN = zeros(1,numel(YTest));
yNN(y == '1') = 1;
yNN(y == '-1') = -1;

%compute NN accuracy on test set
NetACC = sum(yNN' == YTest)/numel(YTest);
```

3.3.2 compare with SVM

```
gamma = 0.005; %SVM hyperparam
[w, b, xi, a] = softsvm(XTrain,YTrain,gamma);

%compute SVM accuracy on test set
ySVM = sign(w'*XTest+b);
SVMACC = sum(ySVM' == YTest)/numel(YTest);

%clearly, neural network with ~3% misclassification rate performed better
%than SVM with ~8% misclassification rate
fprintf('misclassification rate of SVM is: %d \n', 1-SVMACC);
fprintf('misclassification rate of Network is: %d \n', 1-NetACC);
```

Published with MATLAB® R2020b




```
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.

<stopping criteria details>
misclassification rate of SVM is: 8.309456e-02
misclassification rate of Network is: 2.722063e-02
>>
```