Convolutional networks

Table of Contents

Initialization

```
clear; clc;
loadNetworks = true;
```

Load and convert data to the desired format

```
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(3);
imageSize = [28 28 1];
```

Network 1

```
layers1 = [
    imageInputLayer(imageSize)

    convolution2dLayer(5, 20, 'Stride', 1, 'Padding',
 1, 'WeightsInitializer', 'narrow-normal');
    reluLayer

    maxPooling2dLayer(2, 'Stride', 2);

    fullyConnectedLayer(100, 'WeightsInitializer', 'narrow-normal')
    reluLayer

    fullyConnectedLayer(10, 'WeightsInitializer', 'narrow-normal')
    softmaxLayer

    classificationLayer];

options1 = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'MaxEpochs', 60, ...
    'MiniBatchSize', 8192, ...
    'InitialLearnRate',0.001, ...
    'ValidationPatience', 5, ...
    'ValidationFrequency', 30, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', {xValid, tValid}, ...
    'Plots','training-progress');

if ~loadNetworks
    [net1, trainingInfo1] = trainNetwork(xTrain, tTrain, layers1,
 options1);
end
```

Network 2

```
layers2 = [
    imageInputLayer(imageSize)
    convolution2dLayer(3, 20, 'Stride', 1, 'Padding',
 1, 'WeightsInitializer', 'narrow-normal');
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2, 'Stride', 2);

    convolution2dLayer(3, 30, 'Stride', 1, 'Padding',
 1, 'WeightsInitializer', 'narrow-normal');
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2, 'Stride', 2);

    convolution2dLayer(3, 50, 'Stride', 1, 'Padding',
 1, 'WeightsInitializer', 'narrow-normal');
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10, 'WeightsInitializer', 'narrow-normal')
    softmaxLayer
    classificationLayer];

options2 = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'MaxEpochs', 30, ...
    'MiniBatchSize', 8192, ...
    'InitialLearnRate',0.01, ...
    'ValidationPatience', 5, ...
    'ValidationFrequency', 30, ...
    'Shuffle', 'every-epoch', ...
    'ValidationData', {xValid, tValid}, ...
    'Plots','training-progress');
if ~loadNetworks
    [net2, trainingInfo2] = trainNetwork(xTrain, tTrain, layers2,
 options2);
end
```

Save networks

```
if ~loadNetworks

 save('Homework3_1_networks.mat', 'net1', 'trainingInfo1', 'net2', 'trainingInfo2'
end
```

Load networks

```
if loadNetworks
    load('Homework3_1_networks.mat')
end
```

Test network 1

```
trainAccuracy1 = trainingInfo1.TrainingAccuracy(end);
validAccuracy1 = trainingInfo1.ValidationAccuracy(end);
testAccuracy1 = GetTestAccuracy(net1, xTest, tTest)*100;

fprintf('Network 1 results: training accuracy = %.4f, validation
 accuracy = %.4f, and test accuracy = %.4f.\n', ...
    trainAccuracy1, validAccuracy1, testAccuracy1);
```

Test network 2

```
trainAccuracy2 = trainingInfo2.TrainingAccuracy(end);
validAccuracy2 = trainingInfo2.ValidationAccuracy(end);
testAccuracy2 = GetTestAccuracy(net2, xTest, tTest)*100;

fprintf('Network 2 results: training accuracy = %.4f, validation
 accuracy = %.4f, and test accuracy = %.4f.\n', ...
    trainAccuracy2, validAccuracy2, testAccuracy2);
```

Functions

```
function testAccuracy = GetTestAccuracy(network, xTest, tTest)
    yTest = classify(network, xTest);
    testAccuracy = sum(tTest == yTest)/numel(yTest);
end

function [trainAccuracy, validAccuracy, testAccuracy] =
 testNetwork(network, xTrain, tTrain, xValid, tValid, xTest, tTest)
    yTrain = classify(network, xTrain);
    trainAccuracy = sum(tTrain == yTrain)/numel(yTrain);

    yValid = classify(network, xValid);
    validAccuracy = sum(tValid == yValid)/numel(yValid);

    yTest = classify(network, xTest);
    testAccuracy = sum(tTest == yTest)/numel(yTest);
```

```
end
```

Published with MATLAB® R2020b