Deep Learning Using Bayesian Optimization as Hyperparameter Tuning

This exercise shows how to apply Bayesian optimization to deep learning and find optimal network hyperparameters and training options for convolutional neural networks.

Choose Variables to Optimize

Choose which variables to optimize using Bayesian optimization, and specify the ranges to search in. Also, specify whether the variables are integers and whether to search the interval in logarithmic space. Optimize the following variables:

```
optimVars = [
    optimizableVariable('SectionDepth',[1 3],'Type','integer')
    optimizableVariable('InitialLearnRate',[1e-2 1],'Transform','log')
    optimizableVariable('Momentum',[0.8 0.98])];
    %optimizableVariable('L2Regularization',[1e-10 1e-2],'Transform','log')];
```

Perform Bayesian Optimization

Create the objective function for the Bayesian optimizer, using the training and validation data as inputs. The objective function trains a convolutional neural network and returns the classification error on the validation set. This function is defined at the end of this script. Because `bayesopt` uses the error rate on the validation set to choose the best model, it is possible that the final network overfits on the validation set. The final chosen model is then tested on the independent test set to estimate the generalization error. Data is from the previous exercise.

```
ObjFcn = makeObjFcn(xTrain,xTest,label_Train,label_Test);
```

Perform Bayesian optimization by minimizing the classification error on the validation set.

```
BayesObject = bayesopt(ObjFcn,optimVars, ...
    'MaxTime',14*60, ...
    'IsObjectiveDeterministic',false, ...
    'UseParallel',false);
```

```
|=====================================================================================================================
| Iter | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   | SectionDepth | InitialLearn-|     Momentur
|      | result |             | runtime     | (observed)  | (estim.)    |              | Rate         |
|=====================================================================================================================
|    1 | Best   |     0.15348 |      13.161 |     0.15348 |     0.15348 |            3 |     0.058902 |      0.8591
|    2 | Accept |     0.16949 |      21.636 |     0.15348 |      0.1547 |            3 |     0.015911 |      0.9696
|    3 | Accept |     0.22316 |      14.053 |     0.15348 |     0.15712 |            1 |     0.027601 |       0.829
|    4 | Best   |     0.14972 |      12.642 |     0.14972 |     0.14973 |            1 |     0.043581 |      0.9666
|    5 | Best   |     0.14595 |      10.555 |     0.14595 |     0.14595 |            2 |     0.045768 |      0.9625
|    6 | Accept |     0.28531 |      10.621 |     0.14595 |     0.14595 |            2 |      0.21711 |      0.9609
|    7 | Accept |     0.14783 |      10.508 |     0.14595 |     0.14595 |            2 |     0.010001 |      0.9469
|    8 | Accept |     0.43503 |      10.854 |     0.14595 |     0.14613 |            2 |       0.9911 |       0.940
|    9 | Best   |     0.11488 |      11.103 |     0.11488 |     0.11493 |            2 |     0.070365 |      0.9780
|   10 | Accept |     0.19774 |       11.03 |     0.11488 |     0.15207 |            2 |     0.078393 |      0.9799
|   11 | Accept |     0.21751 |      10.758 |     0.11488 |     0.16732 |            2 |     0.046533 |      0.9799
|   12 | Accept |     0.17891 |      10.857 |     0.11488 |     0.16692 |            2 |     0.010065 |      0.9596
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	SectionDepth	InitialLearn-Rate	Momentum
13	Best	0.10829	10.997	0.10829	0.15252	2	0.010052	0.9505
14	Accept	0.12994	10.972	0.10829	0.14655	2	0.010001	0.96122
15	Accept	0.27119	10.633	0.10829	0.14547	2	0.44695	0.94809
16	Accept	0.23164	10.907	0.10829	0.14496	2	0.31323	0.94367
17	Accept	0.14124	11.562	0.10829	0.1431	2	0.012423	0.95481
18	Accept	0.20245	12.938	0.10829	0.15456	2	0.66652	0.94929
19	Accept	0.12712	18.281	0.10829	0.15531	2	0.12381	0.96844
20	Best	0.078154	18.422	0.078154	0.11001	2	0.11076	0.80662
21	Accept	0.16573	11.75	0.078154	0.1325	2	0.10911	0.94023
22	Accept	0.14313	11.83	0.078154	0.11824	2	0.11837	0.94765
23	Accept	0.078154	14.917	0.078154	0.090146	3	0.12348	0.80178
24	Accept	0.20621	8.9339	0.078154	0.1407	1	0.12117	0.80075
25	Accept	0.15819	9.1411	0.078154	0.11081	1	0.16192	0.80075
26	Accept	0.17514	12.703	0.078154	0.14519	3	0.12109	0.85919
27	Accept	0.2194	8.243	0.078154	0.14296	1	0.034236	0.80255
28	Accept	0.28154	8.9059	0.078154	0.1435	1	0.020841	0.80258
29	Accept	0.15725	8.4708	0.078154	0.14405	1	0.099437	0.80331
30	Accept	0.18267	8.8419	0.078154	0.13705	1	0.545	0.80280

Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 364.5757 seconds
Total objective function evaluation time: 356.2276

Best observed feasible point:

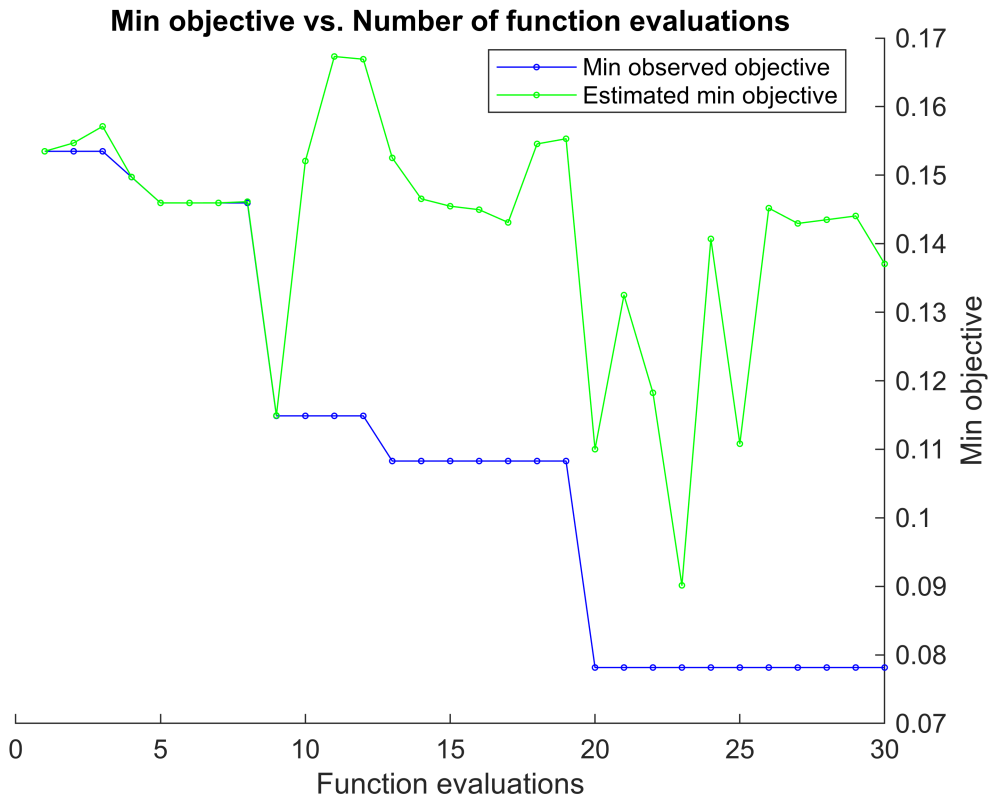
SectionDepth	InitialLearnRate	Momentum
2	0.11076	0.80662

Observed objective function value = 0.078154
Estimated objective function value = 0.13705
Function evaluation time = 18.422

Best estimated feasible point (according to models):

SectionDepth	InitialLearnRate	Momentum
2	0.11076	0.80662

Estimated objective function value = 0.13705
Estimated function evaluation time = 18.1864



Evaluate Final Network

Load the best network found in the optimization and its validation accuracy.

```
bestIdx = BayesObject.IndexOfMinimumTrace(end);
fileName = BayesObject.UserDataTrace{bestIdx};
savedStruct = load(fileName);
valError = savedStruct.valError
```

```
valError = 0.0782
```

```
scoresTest = minibatchpredict(savedStruct.trainedNet,xTest);
YPredicted = onehotdecode(scoresTest,classNames,2);

accuracy = mean(YPredicted == label_Test)
```

```
accuracy = 0.9218
```

Export best selection to ONNX

```
exportONNXNetwork(savedStruct.trainedNet, "hpo_1.onnx");
```

Objective Function for Optimization

Define the objective function for optimization.

```
function ObjFcn = makeObjFcn(XTrain,YTrain,XValidation,YValidation)
ObjFcn = @valErrorFun;
    function [valError,cons,fileName] = valErrorFun(optVars)
        imageSize = [8 8 1];
        numClasses = numel(unique(YValidation));
        classnames = unique(YValidation);
        numF = round(16/sqrt(optVars.SectionDepth));
        layers = [
            imageInputLayer(imageSize)

            % The spatial input and output sizes of these convolutional
            % layers are 8-by-8, and the following max pooling layer
            % reduces this to 4-by-4.
            convBlock(3,numF,optVars.SectionDepth)
            maxPooling2dLayer(1,'Stride',2,'Padding','same')

            % The spatial input and output sizes of these convolutional
            % layers are 16-by-16, and the following max pooling layer
            % reduces this to 8-by-8.
            convBlock(3,2*numF,optVars.SectionDepth)
            maxPooling2dLayer(1,'Stride',2,'Padding','same')

            % The spatial input and output sizes of these convolutional
            % layers are 8-by-8. The global average pooling layer averages
            % over the 8-by-8 inputs, giving an output of size
            % 1-by-1-by-4*initialNumFilters. With a global average
            % pooling layer, the final classification output is only
            % sensitive to the total amount of each feature present in the
            % input image, but insensitive to the spatial positions of the
            % features.
            convBlock(3,4*numF,optVars.SectionDepth)
            averagePooling2dLayer(2)

            % Add the fully connected layer and the final softmax and
            % classification layers.
            fullyConnectedLayer(numClasses)
            softmaxLayer];

        options = trainingOptions("sgdm", ...
            ExecutionEnvironment="cpu", ...
            InitialLearnRate = optVars.InitialLearnRate, ...
            Momentum = optVars.Momentum, ...
            ValidationData={YTrain,YValidation}, ...
            MaxEpochs = 10, ...
            LearnRateSchedule='piecewise', ...
            LearnRateDropPeriod = 40, ...
            LearnRateDropFactor = 0.1, ...
            Verbose=false);
```

```
        imageAugmenter = imageDataAugmenter('RandRotation',[-90,90]);
        datasource =
augmentedImageDatastore(imageSize,XTrain,XValidation,'DataAugmentation',imageAugment
er);

        trainedNet = trainnet(datasource,layers,"crossentropy", options);
        close(findall(groot,'Tag','NNET_CNN_TRAININGPLOT_UIFIGURE'))
        scoresTest = minibatchpredict(trainedNet,YTrain);
        %YPredicted = onehotdecode(scoresTest,int8(classnames),2);
        YPredicted = scores2label(scoresTest,classnames);

        valError = 1 - mean(YPredicted == YValidation);
        fileName = num2str(valError) + ".mat";
        save(fileName,'trainedNet','valError','options')
        cons = [];

    end
end
function layers = convBlock(filterSize,numFilters,numConvLayers)
layers = [
    convolution2dLayer(filterSize,numFilters,'Padding','same')
    batchNormalizationLayer
    reluLayer];
layers = repmat(layers,numConvLayers,1);
end
```