A primer on Deep Learning with Matlab

Multimedia Signal Processing

Starting from Matlab 2017b it is possible to create and test deep networks using the Neural Networks toolbox. In this primer, the main steps required to create a neural network are explained. The creation and testing of a neural network requires the following steps:

- Dataset loading
- Definition of the network architecture
- Definition of the training parameters
- Training and evaluation of the network

1 Dataset loading

In order to load large image datasets which may not completely fit in the available memory, it is necessary to use an ImageDatastore object. To create such object we need the path of the dataset, which in the case of the *DigitDataset* can be obtained as:

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos',
                   'nndatasets','DigitDataset');.
```

At this point, an ImageDatastore object can be created by simply invoking the function

```
imds = ImageDatastore(datasetPath, options).
```

Note that in order to associate each image with its corresponding label it is necessary to add the following options when invoking the `ImageDatastore` functions:

```
'IncludeSubfolders',true,'LabelSource','foldernames'.
```

We can now easily access the dataset content as `imds.Files` and the associated labels as `imds.Labels()`. However, the dataset needs to be partitioned as training and validation datasets. The command used to perform dataset partitioning is the function `splitEachLabel()` which returns two different ImageDatastore objects, one for training and one for validation.

2 Network definition

Defining a neural network means to define the operations which each layer has to perform and how the layers are connected among each other. The architecture can be easily defined as follows:

```
myNet = [ inputLayer
          Layer1
      Activation1
          Layer2
      Activation2
          Layer3
    outputLayer];
```

In Matlab there are several built-in layers which can be used, examples include:

- `imageInputLayer(imageSize)` takes as input an image (`inputLayer`)
- `fullyConnectedLayer(outputSize)` creates a fully connected layer (`Layer`)
- `convolution2dLayer(filterSize,NumOutputFilters)` create a 2d convolutional layer (`Layer`)
- `reluLayer` is the ReLu non-linear activation function
- `softmaxLayer` computes the softmax
- `classificationLayer` assigns a class to the output of the previous layer, computes the crossentropy loss (`outputLayer`)

3 Definition of the training parameters

The algorithms and the parameters which are used to train the network can be set as follows:

```
options = trainingOptions(opts)
```

where some of the available `opts` are

- `'sgdm'` - use stochastic gradient descent *with momentum* (sgdm) as the optimization algorithm
- `'InitialLearnRate',r` - specify the learning rate for the sgdm
- `'Momentum',m` - the momentum parameter of the sgdm (set this parameter to 0 so that the optimization algorithm becomes equivalent to the stochastic gradient descent)
- `'MiniBatchSize',b` - the size b of the mini batch
- `'ValidationFrequency',n` - test the performance of the network on the validation data every n iterations

- `'ValidationData',valData` - specify the dataset on which perform the validation
- `'Plots','training-progress'` - plot the accuracy and loss of the network as the training proceeds
- `'ValidationPatience',Inf` - avoid early stop based on the performance of the network on the validation data
- `'MaxEpochs',e` - number of epochs for the training

4 Training and evaluation of the network

Once the network architecture, its training algortihms and parameters have been set, the network can be trained by using:

```
trainedNet = trainNetwork(imds,myNet,options);
```

The training may take some time depending on the machine, the choice of the parameters and the complexity of the network. When the network has been successfully trained, it can be used to perform the task of interest, e.g. classification. In this case it is possible to input a sample from the dataset and check the output of the network by using:

```
YPred = classify(trainedNet,imds);
```

where `Ypred` contains the prediction of the network for the given input sample.