CUSVM DOCUMENTATION

AUSTIN CARPENTER

cuSVM is a Matlab toolbox for training Gaussian-kernelized (only) Support Vector Machines (SVMs) for either classification or regression. It works by calling CUDA functions via MEX files.

1. Prerequisites

1. A Windows XP 32-bit system.
2. A NVIDIA GT200 or Tesla GPU that supports double precision arithmetic. Geforce 8 or 9 series GPUs will not do unless you modify the source code and compilation settings and then recompile (See Section 3.2.)
3. A working CUDA installation, driver and toolkit. Download at `http://www.nvidia.com/object/cuda_get.html`.
4. A relatively new version of Matlab. The author has tested cuSVM on all releases later than 2007a, inclusive; however, it should work on earlier versions also.

2. Basic Usage

Assuming the prerequisites above are met, all you have to do to use cuSVM is place cuSVMTrain.mexw32 and cuSVMPredict.mexw32, found in the release folder of the cuSVM download, within Matlab's path.

2.1. **cuSVMTrain.** Usage:

[alphas,beta,svs]=cuSVMTrain(y,train,C,kernel,eps, (optional) stoppingcrit)

Outputs:

1. *alphas* is a single-precision vector of the support vector coefficients.
2. *beta* is a single-precision scalar, the offset b in the SVM prediction function.
3. *svs* is a single-precision matrix of the support vectors corresponding to *alphas*, i.e. the support vector found in row i of *svs* has the coefficient in the SVM prediction function found in row i of *alphas*.

Inputs:

1. y is a single-precision vector of training outputs. If you are classifying, these must be either 1 or -1. In regression, these are generally continuously valued.
2. *train* is a single-precision matrix of training data corresponding to y.
3. C is the scalar SVM regularization parameter.
4. *kernel* is the scalar Gaussian kernel parameter, i.e. λ in $\exp(-\lambda \|\mathbf{x}-\mathbf{z}\|^2)$.
5. *eps* is ϵ in ϵ-Support Vector Regression. If you want to classify rather than regress, set *eps* to empty, i.e [ ].

Date: January 14, 2009.

6. *stoppingcrit* is an optional scalar argument that one can use to specify the optimization stopping criterion. By default, the stopping criterion is set to 0.001.

2.2. cuSVMPredict.

Usage:

prediction=cuSVMPredict(test,svs,alphas,beta,kernel,regind)

Outputs:

1. *prediction* is a single-precision vector of predictions.

Inputs:

1. *test* is a single-precision matrix of test data.
2. *svs* is the single-precision matrix of support vectors output by cuSVMTrain.
3. *alphas* is the single-precision vector of support vector coefficients output by cuSVMTrain.
4. *beta* is the single-precision scalar offset output by cuSVMTrain.
5. *kernel* is the same scalar Gaussian kernel parameter value previously used in cuSVMTrain.
6. *regind* is a scalar indicator variable that tells cuSVMPredict whether you are classifying or regressing. Set *regind* to 0 if the former and 1 if the latter.

3. Modifying Source Code

3.1. Basics.

Those who just want to use cuSVM in Matlab can ignore this section.

This, the first, version of cuSVM was developed in CUDA 2.0, which requires that one compile Windows programs using either Microsoft Visual Studio Professional 2005 or Microsoft Visual C++ 2005 Express Edition with the Platform SDK. Both the Express Edition and the Platform SDK can be downloaded for free online (`http://www.microsoft.com/express/2005/download/default.aspx`). See `http://msdn.microsoft.com/en-us/library/ms235626(VS.80).aspx` for instructions on using the Express Edition with the Platform SDK. It is also necessary, whether one uses the Professional or the Express edition, to install the freely downloadable Visual Studio 2005 SP1
(`http://www.microsoft.com/express/2005/download/default.aspx`).

Once you've installed the CUDA driver and toolkit and either VS2005 Pro SP1 or VC++ Express 2005 SP1 with the Platform SDK, you should only have to make two changes in order to build the cuSVM solution:

1. Point VS2005 towards the location of your Matlab installation's MEX libraries. You have to do this for both projects, cuSVMPredict and cuSVMTrain, under both solution configurations, Release and Debug. Select the project in question then go to Project →Properties →Configuration Properties →Linker →General →Additional Library Directories and change

   ```
   D:\Program Files\MATLAB\R2008a\extern\lib\win32\microsoft
   ```

   to the appropriate directory.
2. Point VS2005 towards the location of your Matlab installation's MEX header files. To do this, go to Tools →Options →Projects and Solutions →VC++ Directories → Show directories for: Include files then change

   ```
   D:\Program Files\MATLAB\R2008a\extern\include\
   ```

   to the appropriate directory.

If you are having trouble getting CUDA programs to compile, the NVIDIA CUDA GPU computing forums are a good reference (`http://forums.nvidia.com/index.php?showforum=62`).

Also, all CUDA GPU memory manipulation, and a few other, functions are wrapped in a macro, `mxCUDA_SAFE_CALL`, that when cuSVM is compiled under the Debug solution configuration checks for and reports, using mexPrintf, CUDA errors. This macro, which does nothing when cuSVM is compiled under the Release configuration, is defined in cuSVMutil.h.

3.2. **Strict Single Precision Compilation.** If you don't have a double-precision capable GPU and are confident that a single-precision prediction algorithm is sufficiently accurate for your purposes, you can modify cuSVMPredict so that it uses strict single-precision rather than mixed-precision floating point arithmetic:

1. Go to the `sgemvn_mixedprecis` CUDA function in cuSVMPredictKernel.cu and change the declaration of *sdot* to float rather than double.
2. For both the Debug and Release solution configurations, right click on cuSVMPredictKernel.cu in the Solution Explorer, then go to Custom Build Step →General →Command Line and delete "`--gpu-name sm_13`".

PatternsOnaScreen.net, Apt 18 345 E 12th St., New York, NY 10003