Our SVM training implementation is based on sequential minimal optimization (SMO) algorithm. In SMO algorithm, 2 points are selected from working set and optimised in each iteration. This working set is usually equal to training set. Our implementation uses a working set of specific user-defined size. In one iteration, these steps are performed:

1. Working set selection
   Working set of size N is selected according to the method described below. In the first iteration working set is selected using LibSVM’s first order heuristic.

2. Local kernel matrix calculation
   Kernel matrix tile of size N × N for local solver is computed. If possible, already computed rows are copied from kernel matrix cache.

3. Local solver
   One CUDA kernel uses SMO algorithm to optimize local subproblem. Block size is equal to working set size, because each thread optimizes one element.

4. Kernel cache update
   Rows of kernel cache belonging to alphas which were modified in the local solver are calculated.

5. Gradient update
   Gradient has to be updated only for alphas modified in this iteration.

From our experiments, we found the fastest convergence can be achieved with this working set selection method:

1. Sort all points by how much they violate KKT conditions
2. Pick N/4 first points from both classes
3. Sort points from last iteration’s working set by their score in descending order. Score is a value equal to how many consecutive iterations a particular point was in a working set.
4. Pick enough free vectors to fill new working set
5. If working set is not full, pick enough lower bound vectors to fill new working set
6. If working set is not full, pick enough upper bound vectors to fill new working set.

This working set selection assumes that if a point was selected in working set, it might be selected again in the next iteration. Keeping it in working set for a few iterations takes advantage of already computed kernel values for these points.

Kernel matrix calculation is the most expensive part of the algorithm. Each iteration the kernel values for a few iterations takes advantage of already computed kernel values for these points.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Epison</th>
<th>Alpha</th>
<th>Timit</th>
<th>Adult</th>
<th>Web w8a</th>
<th>MNIST even vs odd</th>
<th>Cosv. Forest</th>
<th>20 Newsgroups</th>
<th>RCv1</th>
<th>Real-Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>0.0001</td>
<td>0.02</td>
<td>0.16</td>
<td>0.02</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Test</td>
<td>0.0001</td>
<td>0.02</td>
<td>0.16</td>
<td>0.02</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>

This work was supported by the Grant Agency of the Czech Republic, project No. GAČR GBP103/12/G084.