GPU-Accelerated SVM Learning for Extremely Fast Large-Scale Proteomics Analysis

John T. Halloran
Department of Public Health Sciences
University of California, Davis
jthalloran@ucdavis.edu

David M. Rocke
Department of Public Health Sciences
University of California, Davis
dmrocke@ucdavis.edu

Abstract

In proteomic analysis pipelines, semi-supervised support vector machine (SVM) learning [12] is a critical step towards accurately identifying the generating peptides of tandem mass spectra. Called Percolator, this algorithm iteratively learns the linear decision boundary between correct and incorrect peptide-spectrum matches (PSMs) and uses the converged decision boundary to rerank the input PSMs. While this reranking greatly improves peptide identification accuracy, Percolator requires substantial analysis time, particularly for the large-scale datasets commonly produced in modern protein studies. Recent work [9] has reduced such lengthy runtimes by updating Percolator's SVM solver to state-of-the-art, multithreaded solvers. In this work, we present primal solvers for linear SVMs novelly designed to take advantage of graphical processing units (GPUs). The resulting GPU solvers drastically reduce Percolator runtime for large-scale analysis, offering up to a 72% improvement over the average performance of previous multithreaded speedups. While implemented within the Percolator codebase, the presented GPU solvers are easily portable to the widely used linear SVM learning packages LIBLINEAR [2] and scikit-learn [18].

1 Introduction

Introduced slightly over a decade ago, semi-supervised SVM learning using the Percolator algorithm [12] has become vital to accurately analyze proteomics data collected via tandem mass spectrometry (MS/MS). Given a collection of MS/MS spectra representing the proteins present in a complex biological sample, the first stage of proteomics analysis typically consists of identifying the input spectra by searching a database of peptides. Database-search thus results in a list of *peptide-spectrum matches* (PSMs). In practice, however, database-search scoring functions are often poorly calibrated, making PSMs from different spectra difficult to compare and diminishing overall identification accuracy. To correct for this, PSM scores are often post-processed using Percolator, which first estimates PSM labels then learns the linear decision boundary between labeled PSMs, repeating these two steps for a user-specified number of iterations. Input PSM scores are subsequently *recalibrated* using the final learned decision boundary.

The accuracy improvements of Percolator recalibration have been well demonstrated for a wide variety of PSM scoring functions (e.g., linear [12, 1, 22], p-value based [4, 10, 15], and dynamic Bayesian networks [6]), complex PSM feature sets (e.g., Fisher kernels [7, 8], subscores of linear functions [19], ensembles of scoring functions [21], and features derived using deep models [3]), and relative to other popular post-processors [20]. Indeed, many complex, state-of-the-art proteomics workflows have adapted Percolator as a critical component of their analysis pipelines. However, while Percolator offers significant accuracy gains, they come at lengthy runtimes as the size of commonly produced proteomic datasets has dramatically increased (by several orders of magnitude) since

14th Machine Learning in Computational Biology (MLCB) Meeting 2019, Vancouver, Canada.

Percolator's debut. For instance, modest datasets comprised of only several million PSMs require several hours of Percolator analysis [9], while more common proteomics datasets–whose PSMs regularly number in the tens-of-millions–may require up to a day (or more) of analysis time [17].

To speed up the lengthy analysis times required of large-scale studies, recent work [9] has updated Percolator's original SVM solver to the state-of-the-art Trust Region Newton (TRON) algorithm [16, 11] and utilized large numbers of compute cores. Optimized for use within Percolator, this multithreaded version of TRON was shown to drastically reduce large-scale analysis time. Herein, we further improve upon the multithreaded TRON speedups by using GPUs to accelerate computation.

We first present a mixed-architecture solver which combines the strengths of both architectures; CPU multithreading for the essential random access components of TRON and a GPU for fast computation in contiguous memory. Compared to the purely multithreaded TRON solver (referred to as TRON CPU), this TRON CPU+GPU solver considerably speeds up SVM learning time for all considered thread configurations; TRON CPU achieves a 4.3 fold speedup (averaged over all considered threads) over Percolator's current SVM learning engine, whereas TRON CPU+GPU is a thread-average **6.4 times faster**. Next, we present a GPU-optimized solver which, rather than adhering to the random-access design of the original TRON algorithm, focuses on device-side (i.e., GPU) compute with as few transactions between device and host (i.e., CPU) as possible. The TRON GPU solver further reduces overall SVM learning time, resulting in a **7.4 fold speedup** and reducing Percolator learning time from 14.4 hours to just 1.9 hours.

2 Semi-supervised SVM Learning for MS/MS Data using Percolator

In practical MS/MS experiments, ground truth labels (i.e., the true peptides responsible for generating each MS/MS spectrum) are not known a priori; indeed, it is the role of the database-search scoring algorithm to identify these generating peptides. In order to assess the confidence of peptide identifications, two peptide databases are typically searched–a *target* database of real peptides and a *decoy* database of permuted target sequences, which we know do not occur in nature–and used to compute the false discovery rate (FDR).

Percolator receives as input both decoy and target PSMs, along with features derived for each PSM. This data is semi-supervised, as we know that decoy PSMs are incorrect identifications (i.e., belong to the negative class), but we are not certain which target PSMs are correct. Each iteration of Percolator thus begins by calculating the target PSMs which achieve a stringent FDR of 0.01% (i.e., are highly confident identifications) and assigning these targets positive training labels. In order to prevent overfitting and improve generalizability, three-fold cross-validation is carried out over three disjoint partitions of the original dataset, followed by further nested cross-validation within each fold [5]. This results in a total of nine unique train and test sets. For each of these training sets, a linear SVM is trained to discriminate between decoys and positive-labeled targets. At the end of each iteration, the separately learned SVM parameters are then merged and used to recalibrate all PSM scores. This process is repeated for a user-specified number of iterations (ten by default).

While this semi-supervised algorithm is robust in practice and widely used throughout the proteomics community, it is also computationally intensive as analysis time is dominated by the iterative training of many SVMs. To combat the extensive Percolator analysis times required of regularly conducted large-scale protein studies, recent work [9] updated Percolator's original SVM solver (called L2-SVM-MFN [13]) to the Trust Region Newton (TRON) algorithm [16], the state-of-the-art solver used in the popular machine learning packages LIBLINEAR [2] and scikit-learn [18]. TRON was optimized within Percolator to utilize multithreading on shared-memory systems and demonstrated to decrease total SVM training time on datasets of several million PSMs. Most importantly, TRON was shown to speedup Percolator without affecting learned SVM parameters, unlike recently proposed random-subsampling approaches [17].

3 Trust Region Newton for Primal SVM Learning

Consider feature vectors $\boldsymbol{x}_i \in \mathbb{R}^n, i = 1, \ldots, l$ and label vector $\boldsymbol{y} \in \{-1, 1\}^l$. Let $X = [\boldsymbol{x}_1 \ldots \boldsymbol{x}_l]^T$, $\mathbf{1}$ denote the indicator function, and $*$ denote element-by-element vector multiplication. For vectors, index-set subscripts denote subvectors and for matrices, pairs of index-set subscripts denote submatrices.

The L2-regularized, L2-SVM primal objective function, which we wish to minimize w.r.t. $\boldsymbol{w}$, is

$$f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}(\max(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i))^2, \tag{1}$$

the gradient of which is $\nabla f(\boldsymbol{w}) = \boldsymbol{w} + 2CX_{I,:}^T(X_{I,:}\boldsymbol{w} - \boldsymbol{y}_I)$, where $I \equiv \{i|1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i > 0\}$ is an index set and the operator : denotes all elements along the corresponding dimension (i.e., all columns in this case). The generalized Hessian of $f(\boldsymbol{w})$ [13] is $\nabla^2 f(\boldsymbol{w}) = \mathcal{I} + 2CX^T DX$, where $\mathcal{I}$ is the identity matrix and D is a diagonal matrix with elements $D_{ii} = \mathbf{1}_{i\in I}$.

Algorithm 1 TRON for L2-SVMs

1: Given w, Δ, and σ_0
2: **while** Not converged **do**
3: Find $\boldsymbol{d}$ in Equation 2 using a conjugate gradient procedure
4: Calculate $\sigma = \frac{f(\boldsymbol{w}+\boldsymbol{d})-f(\boldsymbol{w})}{q(\boldsymbol{d})}$
5: **if** $\sigma > \sigma_0$ **then** $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{d}$, increase trust region Δ.
6: **else** Shrink Δ.
7: **end if**
8: **end while**

The TRON algorithm is detailed in Algorithm 1. At each iteration, given the current parameters $\boldsymbol{w}$ and trust region interval Δ, TRON considers the following quadratic approximation to $f(\boldsymbol{w}+\boldsymbol{d}) - f(\boldsymbol{w})$, $q(\boldsymbol{d}) = \nabla f(\boldsymbol{w})^T\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T\nabla^2 f(\boldsymbol{w})\boldsymbol{d}$, to find a truncated Newton step confined in the trust region by solving

$$\min_{\boldsymbol{d}} q(\boldsymbol{d}) \quad \text{s.t. } \|\boldsymbol{d}\|_2 \leq \Delta. \tag{2}$$

If $q(\boldsymbol{d})$ is close to $f(\boldsymbol{w}+\boldsymbol{d})-f(\boldsymbol{w})$, $\boldsymbol{w}$ is updated to $\boldsymbol{w}+\boldsymbol{d}$ and the trust region interval is increased for the subsequent iteration. Otherwise, $\boldsymbol{w}$ remains unchanged and the trust region interval is shrunk. The conjugate gradient method used to solve Equation 2 involves only a single Hessian-vector product, the structure of which is exploited to avoid loading the entire Hessian into memory; owing to the diagonal form of D, we have $\nabla^2 f(\boldsymbol{w}) = \mathcal{I} + 2CX_{I,:}^T D_{I,I} X_{I,:}$. Thus, for a vector $\boldsymbol{v}$, the Hessian-vector product computed during conjugate gradient descent is $\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2CX_{I,:}^T(D_{I,I}(X_{I,:}\boldsymbol{v}))$, and the algorithm is very efficient overall.

3.1 TRON Implementations

Due to the special forms of the gradient and generalized Hessian (in particular, the derivation and use of I throughout the algorithm), computation in TRON heavily relies on random access. This naturally allows efficient design of the algorithm on a shared-memory, multicore system. As detailed in [14] (and utilized within Percolator in [9]) the computation of $f(\boldsymbol{w})$, $\nabla f(\boldsymbol{w})$, and $\nabla^2 f(\boldsymbol{w})\boldsymbol{v}$ may be efficiently computed across multiple parallel threads using OpenMP. While efficient for multicore architectures, the non-contiguous nature of the algorithm (i.e., I is recomputed every iteration) make designing an efficient GPU implementation far less straightforward; for GPU computing, device-side computation performs best over contiguous memory. Furthermore, large memory transfers between host (i.e., CPU) and device (i.e., GPU) are expensive, hindering approaches where I is first computed then a randomly accessed subset of the data is formed on the host and transferred to the device.

We present two efficient GPU implementations of TRON with complimentary strengths and weaknesses. Written in CUDA, both implementations first load X and $\boldsymbol{y}$ onto the device and, at the start of each iteration, transfer $\boldsymbol{w}$ from host to device. Both solvers also make distinct use of the insight that, on the device-side, prior to computing $f(\boldsymbol{w})$ in each iteration, I may be computed and stored in device memory for future compute. The major operations of each solver are listed in Table 1. The mixed architecture solver, TRON CPU+GPU, utilizes the GPU for heavy lifting before using multithreading for efficient random access after I is computed, utilizes less device-side memory but requires several large data transfers between host and device. The GPU-optimized solver, TRON GPU, performs all intensive computing on the GPU with very few transactions between host and device (only two small transfers from device to host), at the cost of higher device-side memory to compute and store $X_{I,:}$ every iteration.

TRON CPU+GPU	TRON GPU
$\boldsymbol{z} = \boldsymbol{y} * (X\boldsymbol{w})$ is calculated on the device, then copied to the host	$\boldsymbol{z} = \boldsymbol{y} * (X\boldsymbol{w})$ is calculated on the device
$I = \{i : \boldsymbol{z}_i < 1\}$ is calculated on the device transferred to the host. The transfer is interleaved with the device-side computation of $f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}(1 - z_i > 0)^2$	I is calculated on the device, then $f(\boldsymbol{w})$ is computed on the device while the host runs independent, sequential operations
$g(\boldsymbol{w}) = \nabla f(\boldsymbol{w})$ is computed using multithreading as $g(\boldsymbol{w}) = \boldsymbol{w} + 2CX_{I,:}^T(X_{I,:}\boldsymbol{w} - \boldsymbol{y}_I) = \boldsymbol{w} + 2CX_{I,:}^T(\boldsymbol{z}_I * \boldsymbol{y}_I - \boldsymbol{y}_I) = \boldsymbol{w} + 2C\sum_{i\in I} y_i(z_i - 1)\boldsymbol{x}_i.$	On the device, $\hat{\boldsymbol{z}} = \boldsymbol{y}_I * (\boldsymbol{z}_I - 1)$ and $\hat{X} = X_{I,:}$ are computed and stored in device memory. $g(\boldsymbol{w}) = \nabla f(\boldsymbol{w})$ is then computed as $g(\boldsymbol{w}) = \boldsymbol{w} + 2C\hat{X}^T\hat{\boldsymbol{z}}$ and transferred to the host.
Using multithreading, the Hessian-product is calculated on the host as $\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2CX_{I,:}^T(D_{I,I}(X_{I,:}\boldsymbol{v})) = \boldsymbol{v} + 2C\sum_{i\in I}(\boldsymbol{x}_i^T\boldsymbol{v})\boldsymbol{x}_i$	The Hessian-product is computed on the device as $\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + 2C\hat{X}^T(\hat{X}\boldsymbol{v})$ and transferred to the host

Table 1: Major operations of the TRON solvers designed for GPU compute.

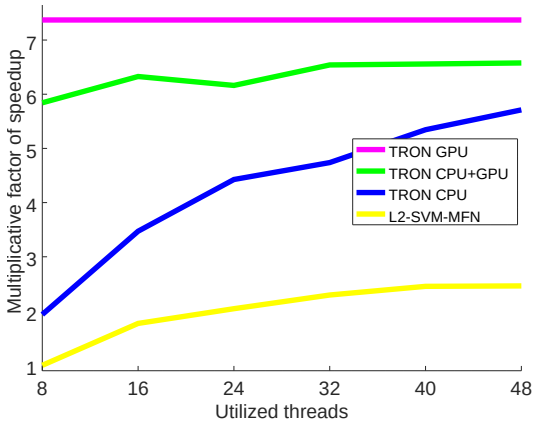

Figure 1: Factor of speedup for SVM learning in Percolator, calculated as the original Percolator SVM learning time divided by the sped up learning time. The x-axis displays the number of threads utilized by multithreaded methods "L2-SVM-MFN," "TRON CPU," and "TRON CPU+GPU."

4 Results and Discussion

All experiments were run on a dual Intel Xeon Gold 5118 compute node with 48 computational threads, an NVIDIA Tesla V100 GPU, and 768 GB of memory. The large-scale dataset evaluated is a larger version of the Kim dataset used in [9] consisting of 23,330,311 PSMs. The GPU optimized TRON solvers are compared against the multithread-optimized versions of TRON and L2-SVM-MFN from [9]. All multithreaded solvers were tested using 8, 16, 24, 32, and 48 threads. As in [9], to effectively measure the runtime of multithreaded methods without any excess thread-scheduling overhead, parallelization of Percolator's outermost cross-validation was disabled. Reported runtimes are the minimum wall-clock times measured over five runs. The original Percolator SVM learning runtime (collected using Percolator v3.02.0) was 14.4 hours. The results are illustrated in Figure 1.

Both GPU solvers greatly accelerate Percolator SVM learning time, with TRON CPU+GPU and TRON GPU completing 6.6 (for 48 threads) and 7.4 times faster than Percolator's current SVM learning engine. Compared to its purely multithreaded counterpart TRON CPU, TRON CPU+GPU offers significantly better performance at low numbers of utilized threads. While TRON GPU offers the best performance overall and the slight memory overhead of the algorithm was nowhere near restrictive on the tested GPU, TRON CPU+GPU offers significant runtime gains for compute environments with more restrictive GPU memory constraints. In future work, the runtime benefits of the presented GPU-optimized solvers will be further evaluated over other large/massive-scale datasets and the memory footprints (on both host and device) will be carefully analyzed.

References

[1] M. Brosch, L. Yu, T. Hubbard, and J. Choudhary. Accurate and sensitive peptide identification with Mascot Percolator. *J. Proteome Res.*, 8(6):3176–3181, 2009.

[2] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[3] Siegfried Gessulat, Tobias Schmidt, Daniel Paul Zolg, Patroklos Samaras, Karsten Schnatbaum, Johannes Zerweck, Tobias Knaute, Julia Rechenberger, Bernard Delanghe, Andreas Huhmer, et al. Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning. *Nature methods*, 16(6):509, 2019.

[4] Viktor Granholm, Sangtae Kim, Jose CF Navarro, Erik Sjolund, Richard D Smith, and Lukas Kall. Fast and accurate database searches with ms-gf+ percolator. *J. Proteome Res.*, pages 890–897, 2013.

[5] Viktor Granholm, William Stafford Noble, and Lukas Kall. A cross-validation scheme for machine learning algorithms in shotgun proteomics. *BMC bioinformatics*, 13(16):S3, 2012.

[6] John T Halloran, Jeff A Bilmes, and William S Noble. Dynamic bayesian network for accurate detection of peptides from tandem mass spectra. *Journal of Proteome Research*, 15(8):2749–2759, 2016.

[7] John T Halloran and David M Rocke. Gradients of generative models for improved discriminative analysis of tandem mass spectra. In *Advances in Neural Information Processing Systems*, pages 5728–5737, 2017.

[8] John T Halloran and David M Rocke. Learning concave conditional likelihood models for improved analysis of tandem mass spectra. In *Advances in Neural Information Processing Systems*, pages 5420–5430, 2018.

[9] John T Halloran and David M Rocke. A matter of time: Faster percolator analysis via efficient svm learning for large-scale proteomics. *Journal of proteome research*, 17(5):1978–1982, 2018.

[10] J Jeffry Howbert and William S Noble. Computing exact p-values for a cross-correlation shotgun proteomics score function. *Molecular & Cellular Proteomics*, pages mcp–O113, 2014.

[11] Chih-Yang Hsia, Ya Zhu, and Chih-Jen Lin. A study on trust region update rules in newton methods for large-scale linear classification. In *Asian Conference on Machine Learning*, pages 33–48, 2017.

[12] L. Kall, J. Canterbury, J. Weston, W. S. Noble, and M. J. MacCoss. A semi-supervised machine learning technique for peptide identification from shotgun proteomics datasets. *Nat. Methods*, 4:923–25, 2007.

[13] S Sathiya Keerthi and Dennis DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6(Mar):341–361, 2005.

[14] Mu-Chu Lee, Wei-Lin Chiang, and Chih-Jen Lin. Fast matrix-vector multiplications for large-scale logistic regression on shared-memory systems. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 835–840. IEEE, 2015.

[15] Andy Lin, J Jeffry Howbert, and William Stafford Noble. Combining high-resolution and exact calibration to boost statistical power: A well-calibrated score function for high-resolution ms2 data. *Journal of proteome research*, 17(11):3644–3656, 2018.

[16] Chih-Jen Lin, Ruby C Weng, and S Sathiya Keerthi. Trust region newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9(Apr):627–650, 2008.

[17] The Matthew, Michael J MacCoss, William S Noble, and Lukas Käll. Fast and accurate protein false discovery rates on large-scale proteomics data sets with percolator 3.0. *Journal of The American Society for Mass Spectrometry*, 27(11):1719–1727, 2016.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] M. Spivak and W. S. Noble. Learning score function parameters for improved spectrum identification in tandem mass spectrometry experiments. *J. Proteome Res.*, 11(9):4499–4508, 2012. PMC3436966.

[20] Chengjian Tu, Quanhu Sheng, Jun Li, Danjun Ma, Xiaomeng Shen, Xue Wang, Yu Shyr, Zhengping Yi, and Jun Qu. Optimization of search engines and postprocessing approaches to maximize peptide and protein identification for high-resolution mass data. *Journal of proteome research*, 14(11):4662–4673, 2015.

[21] Bo Wen, Chaoqin Du, Guilin Li, Fawaz Ghali, Andrew R Jones, Lukas Käll, Shaohang Xu, Ruo Zhou, Zhe Ren, Qiang Feng, et al. Ipeak: An open source tool to combine results from multiple ms/ms search engines. *Proteomics*, 15(17):2916–2920, 2015.

[22] Mingguo Xu, Zhendong Li, and Liang Li. Combining percolator with x! tandem for accurate and sensitive peptide identification. *Journal of proteome research*, 12(6):3026–3033, 2013.