Implementation of Tensor Methods for Computing the 2-Wasserstein Metric

Max Aksel Bowman[1]
[1]Department of Electrical and Computer Engineering
Rice University
Houston, TX 77005
mab31@rice.edu

***Abstract*—The 2-Wasserstein metric quantifies distance between probability distributions with applications in classification, graph comparison, and medical imaging. Computing this distance involves solving an optimal transport problem whose dual can be represented as an unconstrained minimization problem. Typically, first-order approaches are used for minimization problems due to their computational efficiency at the expense of a slow convergence rate. In this report, we present a GPU-accelerated Python library with implementations of first-, second-, and third-order optimization methods for solving this problem. Implementation verification and convergence performance for these methods are discussed. While most first- and second-order methods performed well, we found accelerated Newton and hyperfast methods to converge very slowly or not at all. We hypothesize this is because our Lipschitz constants are not tight. Runtime reductions for logistic regression due to GPU acceleration are demonstrated for second-order methods. Finally, we conclude and discuss future directions of this research.**

***Index Terms*—higher-order methods, computational optimal transport, 2-Wasserstein metric**

I. Introduction

THERE exist several metrics used to compute the notion of a distance between probability distributions. Common choices include Kullback-Leibler divergence, the Kolmogorov-Smirnov statistic, and the infinity-norm difference. Many of these distance metrics suffer from an inability to fully capture key features of the data. Computational optimal transport offers a solution in the form of the 2-Wasserstein distance.

Computing this metric is an optimal transport problem with Kantorovich's relaxation applied to probability distributions. The following presentation of this problem is from [1]. Consider two discrete probability distributions represented as vectors $p, q \in \mathbb{R}^n$. Now, consider integer indices $i, j \in [1, n]$. We can imaging the probabilities plotted along an axis of indices ranging from $[1, n]$. For the purposes of this report, we define the notion of distance between indices i and j to be $(i-j)^2$.

Intuitively, the 2-Wasserstein metric between p and q is the minimal amount of work required to change p into q by moving probability masses along the indices axis. Work in this case is defined as the total sum of the product of masses moved and the distance by which they are moved. Now, let us define this problem formally. Cost matrix $M \in \mathbb{R}^{n \times n}$ is defined as shown in Equation 1. It defines the notion of distance between indices i and j.

$$M_{ij} = (i-j)^2 \tag{1}$$

We define matrix $X \in \mathbb{R}^{n \times n}$ to be the Kantorovich transport plan from p to q. Essentially, X_{ij} describes how much probability mass to move from index i to index j. The primary restriction associated with X is that its sum across rows and columns must match with p and q, respectively. Let $U(p, q)$ denote the space of valid Kantorovich transport plans from p to q. Formally, we are interested in solving Equation 2. We are looking for some valid transport plan that minimizes the sum of the Frobenius dot product between the cost matrix and plan and the entropy of the transport plan. This term is referred to as entropy regularization, and prior literature sets $\gamma = 0.1$.

$$W_\gamma(p, q) = \min_{X \in U(p,q)} \left(\langle M, X \rangle - \gamma \sum_{i,j} X_{ij} \ln(X_{ij}) \right) \tag{2}$$

The definition of the 2-Wasserstein metric presented in Equation 2 is very difficult to solve. There is, however, a dual problem that is much easier to solve [1]. Letting $\lambda^T = [\xi^T \eta^T]$, we can compute $\mathbf{argmin}_{\lambda \in \mathbb{R}^{2n}} l(\lambda)$ where $l(\lambda)$ is the dual problem shown in Equation 3.

$$l(\lambda) = \mathbf{smax}_\gamma(A^T \lambda - \mathbf{vec}(M)) - \lambda^T b \tag{3}$$

The function $\mathbf{smax}_\gamma(x)$ is defined in Equation 4.

$$\mathbf{smax}_\gamma(x) = \gamma \log \left(\sum_{i=1}^{m} e^{x_i/\gamma} \right) \tag{4}$$

Now, we can map the solution from dual space to primal space using Equation 5. In this equation, all exponentials are element-wise exponentials, not matrix exponentials.

$$X(\xi, \eta) = \frac{\mathbf{diag}(e^{\frac{\xi}{\gamma}}) e^{\frac{-M}{\gamma}} \mathbf{diag}(e^{\frac{\eta}{\gamma}})}{e^{\frac{\xi}{\gamma}} e^{\frac{-M}{\gamma}} e^{\frac{\eta}{\gamma}}} \tag{5}$$

In order to solve the minimization problem in Equation 3, we use accelerated higher-order optimization methods. Higher-order methods locally approximate loss functions using second-, third-, and higher-order derivatives. The critical path length for computing higher-order information can be lessened via GPU parallelization, which we discuss in this paper.

In order to use higher-order methods, We define the Lipschitz constant L_p as shown in Equation 6. Note that L_p limits the maximum norm of the difference between p-order derivatives of f.

$$\|\nabla^p f(y) - \nabla^p f(x)\| \leq L_p \|y - x\| \tag{6}$$

The format of this report is as follows. In Section II, we discuss the techniques used in our library to guarantee numerical stability and robustness of our optimization method implementations. We also present our methods for verifying implementation correctness, benchmarking performance, and parallelizing our library. In Section III we present the results associated with each of our research methods. Finally, in Section IV, we conclude and discuss future directions of this work.

II. Methods

A. Optimization Techniques

In this section, I will discuss the optimization techniques implemented in the Hoop3 library. The first of these techniques is the gradient descent, one of the most widely-used and simple to implement methods. Equation 7 describes its update step.

$$x_{k+1} = x_k - \frac{1}{L_1} \nabla f(x_k) \tag{7}$$

Another common first-order technique is the Nesterov accelerated gradient method. The particular implementation we choose is outlined in [2]. The central idea behind this method is to use information from the previous iterate to converge faster. Because the previous iterate is included at each step, we are essentially using old information to balance new information. This approach is shown in Equations 8, 9, and 10.

$$\begin{aligned} y_{k+1} &= x_k - \frac{1}{L_1} \nabla f(x_k) \\ x_{k+1} &= (1 - \gamma_k) y_{k+1} + \gamma_k y_k \end{aligned} \tag{8}$$

$$\lambda_k = \frac{1 + \sqrt{1 + 4\lambda_k^2}}{2} \tag{9}$$

$$\gamma_k = \frac{1 - \lambda_k}{\lambda_{k+1}} \tag{10}$$

Among the oldest of higher-order methods is Newton's method, which minimizes a second-order local approximation at each iterate. This approximation is normally regularized with a quadratic term (see Equation 11).

$$\begin{aligned} \hat{f}_{reg}(x_{k+1}) = f(x_k) + \nabla f(x_k)^T (x_{k+1} - x_k) + \\ (x_{k+1} - x_k)^T \nabla^2 f(x_k)(x_{k+1} - x_k) + \\ \alpha \|x_{k+1} - x_k\|_2^2 \end{aligned} \tag{11}$$

Minimizing this approximation by finding the first-order stationary point results in the Newton update step shown in Equation 12. Note the additional η damping factor, which is a common empirical parameter used to control step size [3] that can be determined via a line search method. We did not use this parameter in our project, choosing to let $\eta = 1$.

$$x_{k+1} = x_k - \eta (\nabla^2 f(x_k) + \alpha I)^{-1} \nabla f(x_k) \tag{12}$$

The cubic Newton method [4] is similar to the Newton method but utilizes a cubic regularization scheme. This means that the same approximation as shown in Equation 11 is used, but with $\frac{M}{6}\|x_{k+1} - x_k\|_2^3$ where $M \geq L_2$ in place of the regularization term.

Accelerated cubic Newton's method [5] is a version of cubic Newton method with a similar acceleration scheme to what is described in first-order. I will not discuss the specifics here, but they can be found in the references.

For the sake of brevity, I will also not discuss the hyperfast method [6] in detail. Effectively, a third-order local Taylor approximation regularized by a quartic penalty is minimized at each iteration. Solving this problem is not as straightforward as in the second-order case because we are now dealing with third-order information. Nesterov uses techniques developed in [7] to iteratively solve this auxiliary problem at each iteration of the hyperfast algorithm. Implementing this algorithm robustly in practice without adaptive methods is difficult because of the difficulty of obtaining tight high-order Lipschitz constants.

B. Numerical Stability Considerations

Numerical instability is a large issue for solving the optimal transport problem. Note that both the gradient of Equation 3 and Equation 5 involve exponentials that can become very large. For example, below is a snippet from naively computing the gradient of Equation 3:

```
result = np.exp(x/gamma)
result /= np.sum(exp_vector)
```

This can be shown to be equivalent to the following computation, which eliminates exponents to powers larger than 1.

```
scaled_x = x/gamma
scaled_x -= scaled_x.max()
exp_vector = np.exp(scaled_x)
result = exp_vector
result /= np.sum(exp_vector)
```

A similar problem appeared while computing the transport plan (see Equation 5). Once again, this equation suffers from poor numerical stability due to its exponential terms. Vectors like ξ/γ, even of moderate size, will cause both the numerator and denominator to blow up, resulting in extremely poor results. In our library, we have addressed this as follows. First, observe that the following equation for the entries of transport plan X follows from Equation 5.

$$X_{ij} = \frac{e^{(-M_{ij} + \xi_i + \eta_j)/\gamma}}{\sum_{i=1}^n \sum_{j=1}^n e^{(-M_{ij} + \xi_i + \eta_j)/\gamma}} \tag{13}$$

Notice that if we define $A_{ij} = -M_{ij} + \xi_i + \eta_j$, we can simply vectorize A and reuse the numerically stable method used for computing optimal transport gradients. This allows

us to compute transport plans even for very light entropy regularization.

C. Verifying Correctness

In [6], a class of functions difficult for all tensor methods is defined. We restate the definition of this class of functions here for clarity. Suppose we define integer parameters n, $p \geq 1$, and $2 \leq k \leq n$. Next, define $U_k \in \mathbb{R}^{k \times k}$ as shown in Equation 14. Now, we can define the block diagonal matrix $A_k \in \mathbb{R}^{n \times n}$ as shown in Equation 15.

$$U_k = \begin{bmatrix} 1 & -1 & 0 & ... & 0 \\ 0 & 1 & -1 & ... & 0 \\ & & \vdots & & \\ 0 & 0 & ... & 1 & -1 \\ 0 & 0 & ... & 0 & 1 \end{bmatrix} \tag{14}$$

$$A_k = \begin{bmatrix} U_k & 0 \\ 0 & I_{n-k} \end{bmatrix} \tag{15}$$

Now, we can define a helper function $\eta_{p+1}(x)$.

$$\eta_{p+1}(x) = \frac{1}{p+1} \sum_{i=1}^{n} |x^{(i)}|^{p+1} \tag{16}$$

Nesterov finally defines the following difficult function for all iterative tensor methods:

$$f_k(x) = \eta_{p+1}(A_k x) - \langle e_1, x \rangle \tag{17}$$

Nesterov further shows that $L_p(\eta_{p+1}(x)) = p!$ and that $\|A_k\|_2 \leq 2$, meaning $L_p(f(x)) \leq 2p!$ Nesterov describes the analytical solution for x^* which minimizes $f_k(x)$. This solution is shown in Equation 18 and has an associated value $f_k(x^*) = -kp/(p+1)$.

$$x^* = [k, k-1, k-2, \ldots, 2, 1, 0, 0, \ldots, 0]^T \tag{18}$$

We can use this class of functions to build confidence that our methods are implemented correctly by checking which value and parameters to which they converge. It is important to use $p = 1$ for first-order methods, $p = 2$ for second-order methods, and $p = 3$ for third-order methods.

We can also use the logistic regression problem for a similar purpose. Consider N data points x_i with associated labels $y_i \in \{-1, 1\}$. Equation 19 shows the loss function for the binary logistic regression problem. By checking the classification accuracy of the solutions to this problem from our implementations, we can verify their correctness.

$$f(\theta) = \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-y_i \mathbf{x_i}^T \theta)) \tag{19}$$

D. Benchmarking Performance

In order to accurately compare the convergence rate of optimization methods, we must ensure that we optimize a function that is not unfairly suited for any one of these methods. In [6], a class of functions difficult for all of these methods are defined. We restate the definition of these functions here for clarity.

E. GPU Parallelization Strategy

The primary bottleneck for higher order methods is computation of the Hessian matrix as well as information derived from the third-order tensor. In this project, our GPU acceleration exploration focused on the logistic regression problem. Initially, we attempted to naively use PyTorch tensors on GPU for all computations. We found, however, that this approach caused the library to run slower. Our next approach was to use Codon, which is a Python-like compiled language with GPU support. Our final solution makes use of Numba [8], which supports JIT compilation and Python specification of CUDA kernels.

Our parallelization strategy for second-order methods solving logistic regression is to assign each Hessian matrix entry to a CUDA threads.

III. Results

We implemented the aforementioned techniques in Hoop3, an extensible Python library with support for GPU acceleration. The library is available on GitHub [9]. It is built with several scientific computing Python libraries, including Numpy [10], Scipy [11], and Numba [8]. Additionally, in early explorations of third-order tensor methods, JAX [12] was used for computing third-order tensors.

A. Lipschitz Constants

Note that $l(\lambda)$ is desirable as a dual due to its property of higher-order smoothness. For this analysis, we can ignore the dot product term because it is linear and therefore does not affect smoothness. Let us denote the p-order Lipschitz constant of the softmax function with L_p. Equation 20 is shown in [13].

$$L_p = \frac{(\frac{p+1}{\ln(p+2)})^{(p+1)} p!}{\mu^p} \tag{20}$$

The function $l(\lambda)$, however, involves an affine transformation inside the argument a softmax.

$$\begin{aligned} &\|\nabla^p \mathbf{smax}_\mu(A^T \lambda_2 - \mathbf{vec}(M)) \\ &-\nabla^p \mathbf{smax}_\mu(A^T \lambda_1 - \mathbf{vec}(M))\| \\ &\leq L_p \|A^T(\lambda_2 - \lambda_1)\| \end{aligned} \tag{21}$$

$$L_p \|A^T(\lambda_2 - \lambda_1)\| \leq L_p \|A^T\| \|\lambda_2 - \lambda_1\| \tag{22}$$

$$L = \frac{(\frac{p+1}{\ln(p+2)})^{(p+1)} p!}{\mu^p} \sigma_0(A) \tag{23}$$

While Lipschitz constants are crucial for ensuring robustness of optimization methods, it is important to note that they are global upper bounds of how much a function or its derivative can change over some amount of distance. Therefore, it is entirely possible that Lipschitz bounds are not tight. Loose bounds will result in very small step sizes, which can cause severe inefficiency in optimization algorithms. It is our current belief that the Lipschitz bounds derived for the optimal transport problem are not very tight, resulting in worse

Application	L_1	L_2	L_3
Log. Reg.	$\frac{1}{4N}\sigma_0^2(X)$	$\frac{1}{6N\sqrt{3}}\max_i \|X_{:,i}\|\sigma_0(X)$	-
Nesterov	2	8	12
OT	$\frac{3.31}{\mu}\sigma_0(A)$	$\frac{20.27}{\mu^2}\sigma_0(A)$	$\frac{228.93}{\mu^3}\sigma_0(A)$

TABLE I
FIRST-, SECOND-, AND THIRD-ORDER LIPSCHITZ CONSTANTS FOR A LOGISTIC REGRESSION, NESTEROV'S DIFFICULT FUNCTIONS FOR TENSOR METHODS, AND OPTIMAL TRANSPORT.

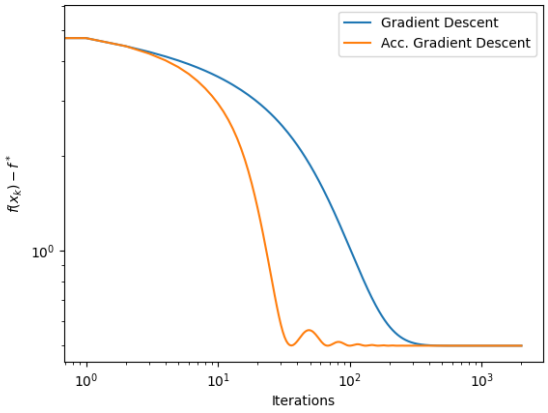

Fig. 1. First-order method convergence for Nesterov's difficult tensor methods ($d = 25$, k=10, $p = 1$).

performance of higher-order methods in comparison to first-order methods for this application.

The Lipschitz constants for Nesterov's difficult tensor functions (with $p = 2$) as well as logistic regression are presented in Table I. L_1 and L_2 for logistic regression are source from [14] and [15], respectively. L_3 is too large to include and is shown in Appendix A. The optimal transport Lipschitz constants are discussed in a previous section.

B. Correctness

The correctness of first- and second-order methods can be verified using Nesterov's difficult tensor functions. Figure 1 shows that the gradient descent and Nesterov accelerated gradient descent implementations are correct. Figure 2 shows that while quadratic- and cubic-regularized Newton implementations are correct, something is incorrect with the accelerated Newton method. It is unclear whether this is truly an issue with the implementation or an issue with the Lipschitz constants used. For both of these graphs, a Lipschitz constant of $2^{1.5}p!$ was used instead of $2p!$.

There are other less direct ways we can verify our implementations. Figure 3 shows an example transport plan for two randomly generated marginals. Visual inspection of the plan passes several sanity checks, such as only the top right quadrant being non-zero. Another simpler sanity check was described in the fall report.

Logistic regression can also be used to verify the correctness of these methods. The classification accuracy for methods besides hyperfast are shown in Table II. The methods are

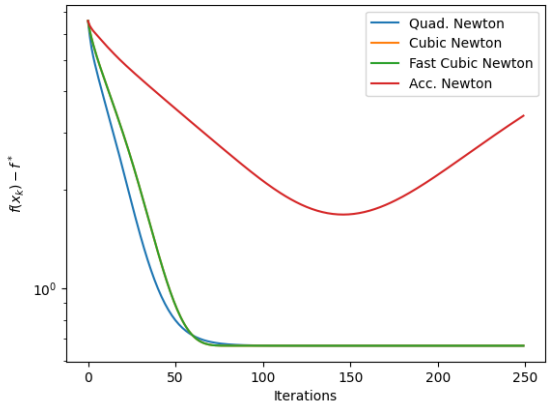

Fig. 2. Second-order method convergence for Nesterov's difficult tensor methods ($d = 25$, k=10, $p = 2$).

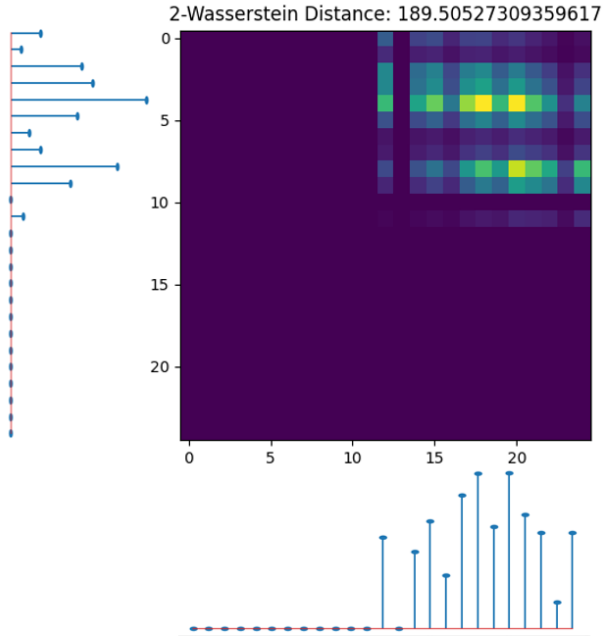

Fig. 3. Kantorovich transport plan and 2-Wasserstein distance computed for two example marginal distributions. This transport plan was obtained using Nesterov accelerated gradient method.

labeled as follows: gradient descent (1), accelerated gradient descent (2), quadratically-regularized Newton method (3), cubic-regularized Newton method(4), and accelerated cubic-regularized Newton method (5). The generally good performance of the resulting binary classifier adds support to the efficacy of the implementations. Note for poorly scaled data, however, the results are poor. Note also that accelerated cubic Newton method can diverge, causing poor results (see the 'australian_scaled' column, for example).

C. Performance

Figures 4 and 5 show the performance of first- and second-order methods for the logistic regression and optimal transport problems. The hyperfast method was only implemented for

Dataset	1	2	3	4	5
a9a	0.837	0.849	0.848	0.849	0.770
australian	0.445	0.445	0.638	0.443/0.434	0.445
australian_scaled	0.875	0.877	0.875	0.880	0.765
ijcnn1	0.903	0.914	0.908	0.915	0.908
mushrooms	0.974	0.999	0.999	1.000	0.995
phishing	0.921	0.937	0.930	0.939	0.915

TABLE II
CLASSIFICATION ACCURACY OF VARIOUS METHODS FOR DIFFERENT LIBSVM DATASETS.

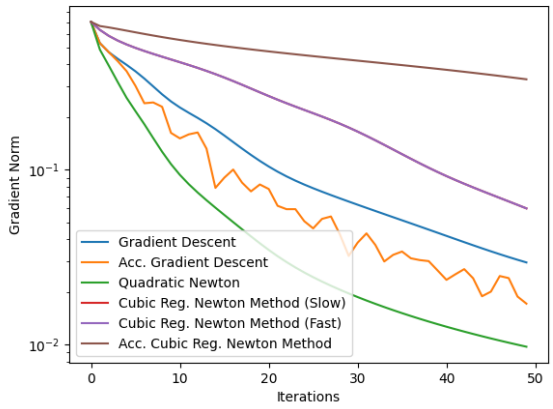

Fig. 4. Gradient norm versus iterations for all methods excluding hyperfast on the optimal transport problem (randomly generated distributions of the form shown in Figure 3 in $S(25)$.

logistic regression, and it did not converge (the loss function value essentially did not change). Note that cubic regularization performs well for the logistic regression case, but not the optimal transport case.

D. GPU Acceleration Performance

All supported second-order methods in Hoop3 were used to benchmark the runtime improvement associated from using the logistic regression Hessian GPU kernel. The results are shown in Figure 6. This figure shows drastic improvements associated

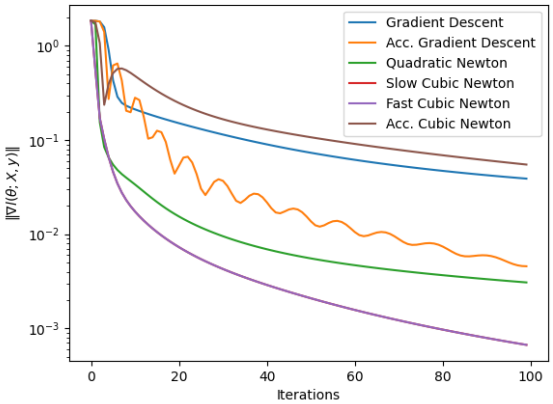

Fig. 5. Gradient norm versus iterations for all methods excluding hyperfast on the LIBSVM mushrooms problem.

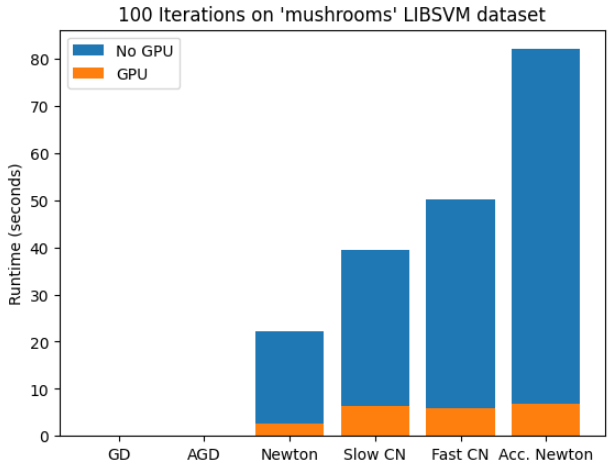

Fig. 6. Runtime improvements second-order methods solving the binary logistic regression problem on the LIBSVM mushrooms dataset. The methods were run for 100 iterations.

with using the GPU, even for an unoptimized kernel that does not utilize advanced GPU concepts such as coalesced memory reads. These results are very promising and motivate future work into using GPUs to reduce the computational burden of higher order methods.

IV. CONCLUSION

In summary, while some higher-order method implementations performed well on optimal transport and logistic regression, others did not. The results of this work make clear the importance of either using extremely tight Lipschitz constants or turning to adaptive methods. Additionally, further examination into the correctness of implementation is warranted. Despite the challenges associated with implementing the accelerated cubic Newton and hyperfast methods, this project has resulted in the creation of an easy-to-use, GPU-accelerated library for higher order methods with supports for both logistic regression and optimal transport problems.

There are several future directions for this research. Our work on this project suggests the investigation of adaptive step size to eliminate issues relating to loose Lipschitz constants. Another direction is the implementation of GPU acceleration for obtaining higher-order information of the optimal transport dual. Yet another interesting direction is the distributed computation of 2-Wasserstein barycenters, which I am investigating in ELEC 570.

ACKNOWLEDGMENT

The author would like to thank Dr. César A. Uribe for advising this research project. The author appreciates the guidance of Dr. Joseph Young, the director of the ECE master's program. The author would also like to thank the Rice Electrical and Computer Engineering department for providing a full tuition scholarship to pursue graduate studies.

REFERENCES

[1] P. Dvurechensky, P. Ostroukhov, A. Gasnikov, C. A. Uribe, and A. Ivanova, "Near-optimal tensor methods for minimizing the gradient norm of convex functions and accelerated primal-dual tensor methods," 2023.

[2] S. Bubeck, "Orf523: Nesterov's accelerated gradient descent." https://web.archive.org/web/20210302210908/https://blogs.princeton.edu/imabandit/2013/04/01/acceleratedgradientdescent/, April 2013.

[3] R. Tibshirani, "Newton's method," 2015.

[4] Y. Nesterov and B. Polyak, "Cubic regularization of newton method and its global performance," *Mathematical Programming*, vol. 108, pp. 177–205, Apr. 2006.

[5] Y. Nesterov, "Accelerating the cubic regularization of newton's method on convex problems," *Mathematical Programming*, vol. 112, p. 159–181, Jan. 2007.

[6] Y. Nesterov, "Implementable tensor methods in unconstrained convex optimization," *Mathematical Programming*, vol. 186, p. 157–183, Nov. 2019.

[7] H. Lu, R. M. Freund, and Y. Nesterov, "Relatively-smooth convex optimization by first-order methods, and applications," 2017.

[8] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: a llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, (New York, NY, USA), Association for Computing Machinery, 2015.

[9] M. A. Bowman, "Higher Order Optimization Library 3 (Hoop3)," April 2024.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[11] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[12] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[13] B. Bullins, "Highly smooth minimization of non-smooth problems," in *Proceedings of Thirty Third Conference on Learning Theory* (J. Abernethy and S. Agarwal, eds.), vol. 125 of *Proceedings of Machine Learning Research*, pp. 988–1030, PMLR, 09–12 Jul 2020.

[14] J. Watt, "6.2 logistic regression and the cross entropy cost*." https://jermwatt.github.io/machine_learning_refined/notes/6_Linear_twoclass_classification/6_2_Cross_entropy.html.

[15] K. Mishchenko, "Regularized newton method with global $o(1/k^2)$ convergence," 2023.

[16] S. (https://math.stackexchange.com/users/15023/stan), "Why is the frobenius norm of a matrix greater than or equal to the spectral norm?." Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/746133 (version: 2015-09-13).

APPENDIX A

In this appendix, we discuss the derivation of L_3 for the logistic regression loss function. First, we start by computing the fourth derivative of $f : \mathbb{R}^d \mapsto \mathbb{R}$, which is shown in Equation 24.

$$\frac{\partial d^4 f}{\partial \theta_w \partial \theta_j \partial \theta_k \partial \theta_l} = \frac{1}{N} \sum_{i=1}^{N} X_{ik} X_{il} X_{ij} X_{iw} g(z_i) \tag{24}$$

$$g(z_i) = \frac{(e^{-z_i} - 3e^{-3z_i})(1 + e^{-z_i}) - 4e^{-z_i}(e^{-z_i} - e^{-3z_i})}{(1 + e^{-z_i})^5} \tag{25}$$

$$z_i = y_i \mathbf{x_i}^T \theta \tag{26}$$

By visual inspection with the Desmos graphing tool, $|g(z_i)| \leq \frac{1}{8}$. Let us denote A as the fourth-order tensor of f. Note that $\|A\|_2 \leq \|A\|_F$ since the induced two-norm of a tensor is less than the Frobenius norm of said tensor (this assumes the matrix case [16] generalizes). Since the induced two-norm of the fourth-order tensor appears to be bounded above by $\|A\|_F$, this bounds the third-order Lipschitz constant of the third-order tensor. Therefore, we arrive at the third-order Lipschitz constant for logistic regression shown in Equation 27.

$$L_3 = \frac{1}{8N} \sqrt{\sum_{w,j,k,l=1}^{d} \sum_{i=1}^{N} (X_{ik} X i l X_{ij} X_{iw})^2} \tag{27}$$

This constant, however, takes a very long to compute, especially if d or N are large, which they usually are. Therefore, we suggest adaptive methods as an alternative.