OPT2022: 14th Annual Workshop on Optimization for Machine Learning

Target-based Surrogates for Stochastic Optimization

Jonathan Lavington* WILDERLAVINGTON@GMAIL.COM
University of British Columbia

Sharan Vaswani* VASWANI.SHARAN@GMAIL.COM
Simon Fraser University

Reza Babanezhad BABANEZHAD@GMAIL.COM
Samsung AI, Montreal

Mark Schmidt SCHMIDTM@CS.UBC.CA
University of British Columbia, Canada CIFAR AI Chair (Amii)

Nicolas Le Roux NICOLAS.LE.ROUX@GMAIL.COM
Microsoft Research

Abstract

We consider minimizing functions for which it is expensive to compute the gradient. Such functions are prevalent in reinforcement learning, imitation learning and bilevel optimization. Our target optimization framework uses the (expensive) gradient computation to construct surrogate functions in a *target space* (e.g. the logits output by a linear model for classification) that can be minimized efficiently. This allows for multiple parameter updates to the model, amortizing the cost of gradient computation. In the full-batch setting, we prove that our surrogate is a global upper-bound on the loss, and can be (locally) minimized using a black-box optimization algorithm. We prove that the resulting majorization-minimization algorithm ensures convergence to a stationary point of the loss. Next, we instantiate our framework in the stochastic setting and propose the `SSO` algorithm that can be viewed as projected stochastic gradient descent in the target space. This connection enables us to use standard stochastic optimization algorithms to construct surrogates which can be minimized using deterministic optimization. Our experiments on supervised learning and imitation learning exhibit the benefits of target optimization, even in stochastic settings.

1. Introduction

Stochastic gradient descent (SGD) [18] and its variants [5, 10] are ubiquitous optimization methods in machine learning (ML). For supervised learning, iterative first-order methods require computing the gradient over individual mini-batches of examples. The computational cost of these algorithms is often dominated by that of computing the stochastic gradient, which can be expensive. For example, in reinforcement learning (RL) or online imitation learning (IL) [19], optimization for finding a good policy requires gathering data samples via potentially expensive interactions with the environment. Policy gradient methods [22, 28]

. $*$ indicates equal contribution.

for example, use the gathered data to compute the stochastic gradient and update the policy. This computation requires expensive interactions with the environment and its cost dominates the total computational cost of these methods.

We focus on algorithms that access the expensive gradient oracle to construct a sequence of surrogate functions. Typically, these surrogates are chosen to be global upper-bounds on the underlying function and hence minimizing the surrogate results in minimizing the function. Algorithmically, these surrogate functions can be minimized efficiently *without additional accesses to the gradient oracle*, making this technique advantageous for the applications of interest. This technique of incrementally constructing and minimizing surrogate functions is commonly referred to as *majorization-minimization* and includes the expectation-maximization (EM) algorithm [4] as an example. In RL, common algorithms [20, 21] also rely on minimizing surrogate functions.

Surrogate functions are typically constructed by using the convexity and/or smoothness properties of the underlying function. Such surrogates have been used in the stochastic setting [13, 14]. Unlike these existing works, we construct surrogate functions over a well-chosen *target space* rather than the parametric space, leveraging the *composition structure* of the loss functions prevalent in ML to build tighter surrogates. For example, in supervised learning, typical loss functions are of the form $h(\theta) = \ell(f(\theta))$, where ℓ is (usually) a convex loss (e.g. the squared loss for regression or the logistic loss for classification), while f corresponds to a transformation (e.g. linear or high-dimensional, and non-convex as in the case of neural networks) of the inputs. Similarly, in IL, ℓ measures the divergence between the policy being learned and the ground-truth expert policy, whereas f corresponds to a specific parameterization of the policy being learned. Formally, if Θ is the feasible set of parameters, $f : \Theta \to \mathcal{Z}$ is a potentially non-convex mapping from the *parametric space* $\Theta \subseteq \mathbb{R}^d$ to the *target space* $\mathcal{Z} \subseteq \mathbb{R}^p$ and $\ell : \mathcal{Z} \to \mathbb{R}$ is a convex loss function. Here, computing $\nabla_z \ell(z)$ requires accessing the expensive gradient oracle, while $\nabla_\theta f(\theta)$ can be computed efficiently. Unlike Nguyen et al. [15] who exploit this composition structure to prove global convergence, we will use it to construct surrogate functions in the target space. The work in Johnson and Zhang [8] also constructs surrogate functions using the target space, but require accessing the (stochastic) gradient oracle for each model update, and is hence inefficient in our setting. We make the following contributions.

Target optimization in the deterministic setting: In Section 3, we use the smoothness of ℓ with respect to z in order to define the *target smoothness surrogate* and prove that it is a global upper-bound on the underlying function h. Using this, we devise a majorization-minimization algorithm which iteratively forms the target smoothness surrogate and (locally) minimizes it using any black-box algorithm. Although forming the target smoothness surrogate requires access to the expensive gradient oracle, it can be minimized without additional oracle calls resulting in multiple, computationally efficient updates to the model. We refer to this framework as *target optimization*. In Algorithm 1 of Appendix B, we instantiate the target optimization framework using m steps of gradient descent to

approximately minimize the surrogate. For smooth surrogates, we prove that Algorithm 1 converges to a stationary point of h at an $O\left(1/T\right)$ rate for *any* value of $m \geq 1$ (Appendix C).

Target optimization in the stochastic setting: In Section 4, we consider the setting where we have access to an expensive stochastic gradient oracle that returns a noisy, but unbiased estimate of the true gradient. Similar to the deterministic setting, we access the gradient oracle to form a *stochastic* target smoothness surrogate. Though the surrogate is constructed by using a stochastic gradient in the target space, it is a deterministic function with respect to the parameters and can be minimized using any standard optimization algorithm. Our framework disentangles the stochasticity in $\nabla_z \ell(z)$ (in the target space) from the potential non-convexity in f (in the parametric space). Similar to the deterministic setting, we use m steps of GD to minimize the stochastic target smoothness surrogate and refer to the resulting algorithm as stochastic surrogate optimization (`SSO`). Interpreting `SSO` as inexact projected SGD in the target space allows us to take advantage of existing stochastic optimization techniques to construct surrogates which themselves can be minimized by a deterministic optimization method.

Minimizing surrogate functions in the target space is also advantageous because it allows us to choose the space in which to constrain the size of the updates. Specifically, for overparameterized models such as deep neural networks, there is only a loose connection between the updates in the parameter and target space. In order to directly constrain the updates in the target space, methods such as natural gradient [1, 9] involve computationally expensive operations. In comparison, `SSO` has direct control over the updates in the target space and can also be implemented efficiently.

Experimental evaluation: To evaluate our target optimization framework, we consider a suite of supervised learning and imitation learning problems (Section 5 and Appendix D), comparing `SSO` with different variations of the target surrogate to standard optimization methods. Our empirical results exhibit the benefits of target optimization.

2. Problem Formulation

We focus on minimizing functions that have a composition structure and for which the gradient computation is expensive. Formally, our objective is to solve the following problem: $\min_{\theta \in \Theta} h(\theta) := \ell(f(\theta))$ where $\Theta \subseteq \mathbb{R}^d$, $\mathcal{Z} \subseteq \mathbb{R}^p$, $f : \Theta \to \mathcal{Z}$ and $\ell : \mathcal{Z} \to \mathbb{R}$. Throughout this paper, we will assume that h is L_θ-smooth[1] in the parameters θ and that $\ell(z)$ is L-smooth in the targets z. For all generalized linear models including linear and logistic regression, $f = X^\top \theta$ is a linear map in θ and ℓ is convex in z. For neural networks, it is typical for the function f mapping X to y to be non-convex but for the loss ℓ to be convex. For RL, the target space is the space of policies, and ℓ is the cumulative loss when using a policy $\pi := f(\theta)$ parameterized by θ. Even though our algorithmic framework can handle non-convex ℓ and f, depending on the specific setting, our theoretical results will assume that ℓ (or h) is (strongly)-convex in θ and f is an affine map.

1. For definitions of smoothness, convexity, and strong-convexity see Appendix A.

For our applications of interest, computing $\nabla_z \ell(z)$ is computationally expensive, whereas $f(\theta)$ (and its gradient) can be computed efficiently. For example, in RL, computing the cumulative loss ℓ (and the corresponding gradient $\nabla_z \ell(z)$) for a policy involves evaluating it in the environment. Since this operation involves interactions with the environment or a simulator, it is computationally expensive. On the other hand, the cost of computing $\nabla_\theta f(\theta)$ only depends on the policy parameterization and does not involve additional interactions with the environment. This structure is also satisfied by online imitation learning that we consider in Section 5. In some cases, it is more natural to consider access to a *stochastic* gradient oracle that returns a noisy, unbiased gradient $\nabla\tilde{\ell}(z)$ such that $\mathbb{E}[\nabla\tilde{\ell}(z)] = \nabla\ell(z)$. We consider the effect of stochasticity in Section 4.

If we do not take advantage of the composition structure nor explicitly consider the cost of the gradient oracle, iterative first-order methods such as GD or SGD can be directly used to minimize $h(\theta)$. At iteration $t \in [T]$, the *parametric GD* update is: $\theta_{t+1} = \theta_t - \eta \nabla h_t(\theta_t)$ where η is the step-size to be selected or tuned according to the properties of h. Since h is L_θ-smooth, each iteration of parametric GD can be viewed as exactly minimizing the quadratic surrogate function derived from the smoothness condition with respect to the parameters. Specifically, $\theta_{t+1} := \arg\min g_t^\theta(\theta)$ where g_t^θ is the *parametric smoothness surrogate*: $g_t^\theta := h(\theta_t) + \langle \nabla h(\theta_t), \theta - \theta_t \rangle + \frac{1}{2\eta} \|\theta - \theta_t\|_2^2$. Minimizing the global upper-bound results in descent on h since $h(\theta_{t+1}) \le g_t^\theta(\theta_{t+1}) \le g_t^\theta(\theta_t) = h_t(\theta_t)$. Similarly, the *parametric SGD* update consists of accessing the stochastic gradient oracle to obtain $\left(\tilde{h}(\theta), \nabla\tilde{h}(\theta)\right)$ such that $\mathbb{E}[\tilde{h}(\theta)] = h(\theta)$ and $\mathbb{E}[\nabla\tilde{h}(\theta)] = \nabla h(\theta)$, and iteratively constructing the *stochastic parametric smoothness surrogate* $\tilde{g}_t^\theta(\theta)$. Specifically, $\theta_{t+1} = \arg\min \tilde{g}_t^\theta(\theta)$, $\tilde{g}_t^\theta(\theta) := \tilde{h}(\theta_t) + \langle \nabla\tilde{h}(\theta_t), \theta - \theta_t \rangle + \frac{1}{2\eta_t} \|\theta - \theta_t\|_2^2$. Here, η_t is the iteration dependent step-size that is decayed according to the properties of h [18]. In contrast to these methods, in the next section, we exploit the smoothness of the losses with respect to the target space and propose a majorization-minimization algorithm in the deterministic setting.

3. Deterministic setting

We consider minimizing $\ell(f)$ in the deterministic setting where we can exactly evaluate the gradient $\nabla_z \ell(z)$. Similar to the parametric case in Section 2, we use the smoothness of $\ell(z)$ w.r.t the target space and define the *target smoothness surrogate* around z_t as: $\ell(z_t) + \langle \nabla_z \ell(z_t), z - z_t \rangle + \frac{1}{2\eta} \|z - z_t\|_2^2$, where η is the step-size to be determined theoretically. Since $z = f(\theta)$, the surrogate can be expressed as a function of θ: $g_t^z(\theta) = \ell(z_t) + \langle \nabla_z \ell(z_t), f(\theta) - z_t \rangle + \frac{1}{2\eta} \|f(\theta) - z_t\|_2^2$, which is in general not quadratic in θ. Similar to the parametric smoothness surrogate, we see that $h(\theta_t) = \ell(f(\theta_t)) = g_t^z(f(\theta_t))$ and if ℓ is L-smooth w.r.t the target space $\mathcal{Z}$, then for $\eta \le \frac{1}{L}$, $g_t^z(f(\theta)) \ge \ell(f(\theta)) = h(\theta)$ for all θ i.e. the surrogate is a global upper-bound on h. Since g_t^z is a global upper-bound on h, similar to GD, we can minimize h by minimizing the surrogate at each iteration i.e. $\theta_{t+1} = \arg\min_\theta g_t^z(\theta)$. However, unlike GD, in general, there is no closed form solution for the minimizer of g_t^z, and we will consider minimizing it approximately. This results in the following meta-algorithm:

for each $t \in [T]$, at iterate θ_t, form the surrogate g_t^z and compute θ_{t+1} by (approximately) minimizing $g_t^z(\theta)$. This meta-algorithm enables the use of any black-box algorithm to minimize the surrogate at each iteration. In Algorithm 1, we instantiate this meta-algorithm by minimizing $g_t^z(\theta)$ using $m \geq 1$ steps of gradient descent. For $m = 1$, Algorithm 1 results in the following update: $\theta_{t+1} = \theta_t - \alpha \nabla g_t^z(\theta_t) = \theta_t - \alpha \nabla h(\theta_t)$, and is thus equivalent to parametric GD with step-size α. Instantiating Algorithm 1 for linear regression with $m = 1$ recovers parametric GD on the least squares objective. On the other hand, minimizing the surrogate exactly (corresponding to $m = \infty$) to compute θ_{t+1} results in the following update: $\theta_{t+1} = (X^\intercal X)^{-1} [X^\intercal (X\theta_t - y)]$ and recovers the Newton update in the parameter space. Hence, for linear regression, approximately minimizing g_t^z using $m \in (1, \infty)$ steps of GD interpolates between a first and second-order method.

In Theorem 1 in Appendix C, we prove that Algorithm 1 with any value of $m \geq 1$ and appropriate choices of α and η results in an $O(1/T)$ convergence to a stationary point of h. Importantly, this result only relies on the smoothness of ℓ and g_t^z, and does not require either $\ell(z)$ or $f(\theta)$ to be convex. Hence, this result holds even when using a non-convex model like deep neural networks, or for problems with non-convex loss function such as in RL. The idea of constructing surrogates in the target space has been recently explored in the context of designing efficient off-policy algorithms for reinforcement learning [25]. Next, we consider the stochastic setting where we only obtain a noisy (though unbiased) gradient estimate.

4. Stochastic setting

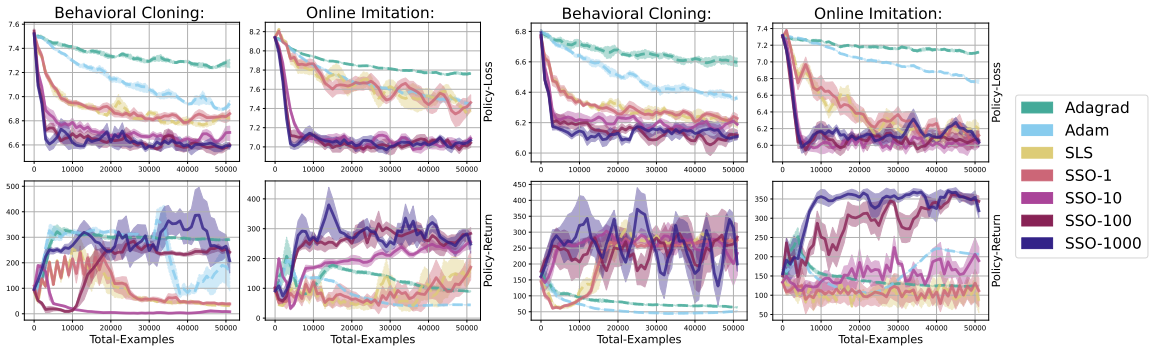
In the stochastic setting, we use the noisy but unbiased estimates $(\tilde{\ell}(z), \nabla\tilde{\ell}(z))$ from the gradient oracle to construct the *stochastic target surrogate*. We will focus on the special case where ℓ is a sum over (potentially infinite) individual losses i.e. $\ell(z) = \sum_i \ell_i(z)$. In this case, querying the stochastic gradient oracle at iteration t returns the individual loss and gradient corresponding to the loss index i_t i.e. $\left(\tilde{\ell}(z), \nabla\tilde{\ell}(z)\right) = (\ell_{i_t}(z), \nabla\ell_{i_t}(z))$. This structure is present in the use-cases of interest, for example in supervised learning using a dataset of n training points or in imitation learning when a small number of random trajectories were collected at iteration t. Also, in order to admit an efficient implementation of the surrogate and the resulting algorithms, we only consider loss functions that are separable w.r.t the target space, i.e. for $z \in \mathcal{Z}$, if $z^i \in \mathbb{R}$ denotes coordinate i of z, then $\ell(z) = \sum_i \ell_i(z^i)$. Such a structure is present in the loss functions for all supervised learning problems where $\mathcal{Z} \subseteq \mathbb{R}^n$ and $z^i = f_i(\theta) := f(X_i, \theta)$. In this setting, $\frac{\partial \ell_i}{\partial z^j} = 0$ for all i and $j \neq i$. The stochastic target surrogate $\tilde{g}_t^z(\theta) = \ell_{i_t}(z_t) + \frac{\partial \ell_{i_t}(z_t)}{\partial z^{i_t}} [f_{i_t}(\theta) - z_t^i] + \frac{1}{2\eta_t} \left[f_{i_t}(\theta) - z_t^{i_t}\right]^2$, with η_t the step-size at iteration t. The next iterate is obtained as $\theta_{t+1} = \arg\min_\theta \tilde{g}_t^z(\theta)$. Note that $\tilde{g}_t^z(\theta)$ only depends on a single point i_t and only requires access to $\partial\ell_{i_t}(z_t)$ and thus can be constructed efficiently.[2] Algorithmically, we can form the surrogate $\tilde{g}_t^z$ at iteration t and minimize it approximately by using any black-box algorithm. Similar to the deterministic setting, we can

2. While $\mathbb{E}[\tilde{g}_t^z] = g_t^z$, $\mathbb{E}[\arg\min \tilde{g}_t^z] \neq \arg\min g_t^z$, in contrast to parametric SGD where $\mathbb{E}[\arg\min \tilde{g}_t^\theta] = \arg\min g_t^\theta$.

minimize $\tilde{g}_t^z$ using m steps of GD. The resulting algorithm is the same as Algorithm 1 but using $\tilde{g}_t^z$. Notably the surrogate is formed by selecting function ℓ_t randomly and is therefore random. However, once the surrogate is formed, it can be minimized using any deterministic algorithm, i.e. there is no additional randomness in the inner-loop in Algorithm 1. Moreover, for $m = 1$, Algorithm 1 has the same update as parametric SGD.

We interpret minimizing $\tilde{g}_t^z$ as projected SGD: $z_{t+1/2} = z_t - \eta_t \nabla_z \ell_{i_t}(z_t)$ and $\bar{z}_{t+1} = \arg\min_{z \in \mathcal{Z}} \frac{1}{2} \left\| z_{t+1/2} - z \right\|_{\mathcal{P}_t}^2$, where $\mathcal{P}_t \in \mathbb{R}^{p \times p}$ is a diagonal preconditioner such that $\mathcal{P}_{i_t, i_t} = 1$ and $\mathcal{P}_{i,j} = 0$ for all $i, j \neq i_t$. Starting from z_t, the SGD update might result in an $z_{t+1/2}$ such that $z_{t+1/2} \notin \mathcal{Z}$, and hence we need an additional projection step. For the linear parameterization, the set $\mathcal{Z}$ is convex, and the Euclidean projection is unique. In Appendix C.1, we prove that $\bar{z}_{t+1} = f\left(\arg\min_{\theta \in \Theta} \tilde{g}_t^z(\theta)\right)$. Hence, minimizing $\tilde{g}_t^z$ is equivalent to projected SGD, implying that the inexact minimization of $\tilde{g}_t^z$ (for example, using m steps of GD in Algorithm 1) can be interpreted as an inexact projection. For non-convex f, the set $\mathcal{Z}$ can be non-convex and the projection is not well-defined but the surrogate $\tilde{g}_t^z$ can still be minimized, albeit without any guarantees on the convergence. To fully instantiate the stochastic surrogate, we can specify the step-size sequence $\{\eta_t\}_{t=1}^T$ using the existing literature [5, 12, 18, 24]. Moreover, while we introduced the framework using a first order method in the target space, it allows us to use alternative stochastic optimization algorithms (e.g online Newton method, stochastic mirror descent) to construct surrogates (refer to Appendix B) that can then be optimized using a black-box deterministic algorithm.

5. Experimental Evaluation



(a) Walker2d-v2 (b) Hopper-v2

Figure 1: Comparison of policy return, and log policy loss incurred by `SGD`, `SLS`, `Adam`, `Adagrad`, and `SSO` as a function of the interactions. In all settings, (i) `SSO` outperforms all other online-optimization algorithms, and (ii) as m in increases `SSO` performance improves. Additional neural network experiments are included in Appendix D, which display similar trends.

We evaluate the target optimization framework for supervised learning and online imitation learning. We compare stochastic surrogate optimization (`SSO`), against `SGD` with

the standard theoretical $1/2L_\theta$ step-size, `SGD` with the step-size set according to a stochastic line-search [24] `SLS`, Adagrad [5], and finally `Adam` [10] using default hyper-parameters. Since `SSO` is equivalent to projected SGD in the target space, we set η (in the surrogate definition) to $1/2L$ where L is the smoothness of ℓ w.r.t z. In the subsequent experiments, we use either the theoretically chosen step-size when available, or the default step-size provided by Paszke et al. [17]. For `SSO`, since the surrogate optimization is a deterministic problem, we use the standard back-tracking Armijo line-search [2] with the same hyper-parameters across all experiments. For each experiment, we plot the average (across 3 independent runs) loss against the number of calls to the (stochastic) gradient oracle. We consider imitation learning below, and defer the results for supervised learning to Appendix D. In imitation learning, the losses are generated through interaction with a simulated environment. In this setting, a behavioral policy gathers examples by observing a state and taking an action at that state. For each state gathered through the interaction, an expert policy provides the action that it would have taken. The goal in IL is to produce a policy which exactly matches that of the expert. When the behavioral policy chosen is the expert itself, we refer to the problem as *behavioral cloning*. When the learned policy is used to interact with the environment [6], we refer to the problem as *online imitation learning* (OIL) [11, 19].

In Fig. 1, we consider continuous control environments from the Mujoco benchmark suite [23]. The policy is a linear function of state (coresponding to f) and parameterizes the mean of a standard normal distribution. Here, the loss (ℓ) is defined as the average squared ℓ_2 distance between the mean action of the policy and the label provided by the expert. During environment interactions for data collection, we sample from the stochastic (multivariate normal) policy in order to take actions. At each round of environment interaction 1000 states are gathered, and used to update the policy. The expert policy, defined by a squashed normal distribution [17], parameterized by a two layer multi-layer-perception, and is trained using the Soft-Actor-Critic Algorithm [7]. A "good" policy in this control setting, is one which maximizes the return. In Fig. 1, we observe the importance of making multiple model updates using one stochastic gradient from the oracle. More specifically, we observe (i) all variants of `SSO` outperform the algorithms by a significant margin in terms of both policy return and log policy loss as a function of environment interactions (calls to the gradient oracle), and (ii) as m increases, the performance of the learned policy improves consistently.

6. Discussion

In the future, we aim to prove convergence rates for the stochastic case, extend our theoretical results to a broader class of functions, and empirically evaluate target optimization for more complex models. Moreover, since our framework allows using any optimizer in the target space, we plan to explore other optimization algorithms to construct better surrogates.

7. Acknowledgements

This research was partially supported by the Canada CIFAR AI Program, the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants RGPIN-2022-03669 and RGPIN-2022-04816.

References

[1] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 1998.

[2] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[3] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[4] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011.

[6] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 158–168. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/florence22a.html.

[7] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018. URL https://arxiv.org/abs/1812.05905.

[8] Rie Johnson and Tong Zhang. Guided learning of nonconvex models through successive functional gradient optimization. In *International Conference on Machine Learning*, pages 4921–4930. PMLR, 2020.

[9] Sham M. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, 2001.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[11] Jonathan Wilder Lavington, Sharan Vaswani, and Mark Schmidt. Improved policy optimization for online imitation learning. *arXiv preprint arXiv:2208.00088*, 2022.

[12] Xiaoyu Li, Zhenxun Zhuang, and Francesco Orabona. A second look at exponential and cosine step sizes: Simplicity, adaptivity, and performance. In *International Conference on Machine Learning*, pages 6553–6564. PMLR, 2021.

[13] Julien Mairal. Stochastic majorization-minimization algorithms for large-scale optimization. *Advances in Neural Information Processing Systems*, 26, 2013.

[14] Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25(2):829–855, 2015.

[15] Lam M Nguyen, Trang H Tran, and Marten van Dijk. Finite-sum optimization: A new perspective for convergence to a global solution. *arXiv preprint arXiv:2202.03524*, 2022.

[16] Francesco Orabona. A modern introduction to online learning. *arXiv preprint arXiv:1912.13213*, 2019.

[17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[18] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[19] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of machine learning research*, pages 627–635, 2011.

[20] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[22] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1057–1063, 2000.

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[24] Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Advances in Neural Information Processing Systems*, pages 3727–3740, 2019.

[25] Sharan Vaswani, Olivier Bachem, Simone Totaro, Robert Mueller, Matthieu Geist, Marlos C Machado, Pablo Samuel Castro, and Nicolas Le Roux. A functional mirror ascent view of policy gradient methods with function approximation. *arXiv preprint arXiv:2108.05828*, 2021.

[26] Sharan Vaswani, Benjamin Dubois-Taine, and Reza Babanezhad. Towards noise-adaptive, problem-adaptive (accelerated) stochastic gradient descent. In *International Conference on Machine Learning*, pages 22015–22059. PMLR, 2022.

[27] Rachel Ward, Xiaoxia Wu, and Leon Bottou. Adagrad stepsizes: Sharp convergence over nonconvex landscapes. *The Journal of Machine Learning Research*, 21(1):9047–9076, 2020.

[28] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Organization of the Appendix

Appendix A. Definitions

Our main assumptions are that each individual function f_i is differentiable, has a finite minimum f_i^*, and is L_i-smooth, meaning that for all v and w,

$$f_i(v) \leq f_i(w) + \langle \nabla f_i(w),\, v - w \rangle + \frac{L_i}{2} \|v - w\|_2^2 \,, \qquad \text{(Individual Smoothness)}$$

which also implies that f is L-smooth, where L is the maximum smoothness constant of the individual functions. A consequence of smoothness is the following bound on the norm of the stochastic gradients,

$$\|\nabla f_i(w) - \nabla f_i^*\|^2 \leq 2L(f_i(w) - f_i^* - \langle \nabla f_i^*, w - w_i^* \rangle). \tag{1}$$

We also assume that each f_i is convex, meaning that for all v and w,

$$f_i(v) \geq f_i(w) + \langle \nabla f_i(w),\, v - w \rangle, \qquad \text{(Convexity)}$$

Depending on the setting, we will also assume that f is μ strongly-convex, meaning that for all v and w,

$$f(v) \geq f(w) + \langle \nabla f(w),\, v - w \rangle + \frac{\mu}{2} \|v - w\|_2^2 \,, \qquad \text{(Strong Convexity)}$$

Appendix B. Algorithms

Algorithm 1 Generic algorithm for surrogate optimization

Input: θ_0 (initialization), T (number of iterations), m_t (number of inner-loops), η (step-size in the surrogate), α (step-size for the parametric update)

for $t \leftarrow 0$ **to** $T - 1$ **do**
- Access the gradient oracle to construct $g_t^z(\theta)$
- Initialize inner-loop: $\omega_0 = \theta_t$
- **for** $k \leftarrow 0$ **to** m_t **do**
  - $\omega_{k+1} = \omega_k + \alpha \nabla_\omega g_t^z(\omega_k)$
- **end**
- $\theta_{t+1} = \omega_m \quad ; \quad z_{t+1} = f(\theta_{t+1})$

end

Return θ_T

In this section, we first specify the generic algorithm for target optimization in the deterministic setting Algorithm 1. Next, we will formulate the algorithms beyond the standard SGD update in the target space. We will do so in two ways – (i) extending SGD to the online Newton step that uses second-order information in Appendix B.1 and (ii) extend SGD to the more general stochastic mirror descent algorithm in Appendix B.2. For both (i) and (ii), we will instantiate the resulting algorithms for the squared and logistic losses.

B.1. Online Newton Step

Let us consider the online Newton step w.r.t to the targets. The corresponding update is:

$$z_{t+1/2} = z_t - \eta_t [\nabla_z^2 \ell_t(z_t)]^{-1} \nabla_z \ell_t(z_t) \quad ; \quad \bar{z}_{t+1} = \arg\min_{z \in \mathcal{Z}} \frac{1}{2} \left\| z - z_{t+1/2} \right\|_{\mathcal{P}_t}^2 \tag{2}$$

$$z_{t+1} = f(\theta_{t+1}) \quad ; \quad \theta_{t+1} = \arg\min_{\theta} \left[\langle \nabla_z \ell_t(z_t), f(\theta) - z_t \rangle + \frac{1}{2\eta_t} \left\| f(\theta) - z_t \right\|_{\nabla^2 \ell_t(z_t)}^2 \right] \tag{3}$$

where $\nabla_z^2 \ell_t(z_t)$ is the Hessian of example of the loss corresponding to sample i_t w.r.t z. Let us instantiate this update for the squared-loss. In this case, $\ell_t(z) = \frac{1}{2} \left\| z - y_t \right\|_2^2$, and hence, $\nabla \ell_t(z) = z - y_t$, $[\nabla^2 \ell_t(z)]_{i_t, i_t} = 1$ and $[\nabla^2 \ell_t(z)]_{j,j} = 0$ for all $j \neq i_t$. Hence, for the squared loss, Eq. (2) is the same as GD in the target space.

For the logistic loss, $\ell_t(z) = \log\left(1 + \exp\left(-y_t z\right)\right)$. If i_t is the loss index sampled at iteration t, then, $[\nabla \ell_t(z)]_j = 0$ for all $j \neq i_t$. Similarly, all entries of $\nabla^2 \ell_t(z)$ except the $[i_t, i_t]$ are zero.

$$[\nabla \ell_t(z)]_{i_t} = \frac{-y_t}{1 + \exp(y_t\, z_t)} \quad ; \quad [\nabla^2 \ell(z)]_{i_t, i_t,} = \frac{1}{1 + \exp(y_t z_t)} \frac{1}{1 + \exp(-y_t z_t)} = (1 - p_t)\, p_t \,,$$

where, $p_t = \frac{1}{1 + \exp(-y_t z_t)}$ is the probability of classifying the example i_t to have the $+1$ label. In this case, the surrogate can be written as:

$$\tilde{g}_t^z(\theta) = \frac{-y_t}{1 + \exp(y_t\, z_t)} \left(f_{i_t}(\theta) - z_t^{i_t} \right) + \frac{(1 - p_t)\, p_t}{2\eta_t} \left(f_{i_t}(\theta) - z_t^{i_t} \right)^2 \tag{4}$$

As before, the above surrogate can be implemented efficiently.

B.2. Stochastic Mirror Descent

If ϕ is a differentiable, strictly-convex mirror map, it induces a Bregman divergence between x and y: $D_\phi(y, x) := \phi(y) - \phi(x) - \langle \nabla \phi(x), y - x \rangle$. For an efficient implementation of stochastic mirror descent, we require the Bregman divergence to be separable, i.e. $D_\phi(y, x) = \sum_{j=1}^p D_{\phi_j}(y^j, x^j) = \sum_{j=1}^p \phi_j(y^j) - \phi_j(x^j) - \frac{\partial \phi_j(x)}{\partial x^j}[y^j - x^j]$. Such a separable structure is satisfied when ϕ is the Euclidean norm or negative entropy. We define stochastic mirror descent update in the target space as follows,

$$\nabla \phi(z_{t+1/2}) = \nabla \phi(z_t) - \eta_t \nabla_z \ell_t(z_t) \quad ; \quad \bar{z}_{t+1} = \arg\min_{z \in \mathcal{Z}} \sum_{j=1}^p \mathbb{I}(j = i_t) D_{\phi_j}(z^j, z_{t+1/2}^j)) \tag{5}$$

$$\implies \bar{z}_{t+1} = \arg\min_{z \in \mathcal{Z}} \left[\langle \nabla_z \ell_t(z_t), z - z_t \rangle + \frac{1}{\eta_t} \sum_{j=1}^{p} \mathbb{I}(j = i_t) D_{\phi_j}(z^j, z^j_{t+1/2}) \right] \tag{6}$$

where $\mathbb{I}$ is an indicator function and i_t corresponds to the index of the sample chosen in iteration t. For the Euclidean mirror map, $\phi(z) = \frac{1}{2}\|z\|_2^2$, $D_\phi(z, z_t) = \frac{1}{2}\|z - z_t\|_2^2$ and we recover the SGD update.

Another common choice of the mirror map is the negative entropy function: $\phi(x) = \sum_{i=1}^{K} x^i \log(x^i)$ where x^i is coordinate i of the $x \in \mathbb{R}^K$. This induces the (generalized) KL divergence as the Bregman divergence,

$$D_\phi(x, y) = \sum_{k=1}^{K} x^k \log\left(\frac{x^k}{y^k}\right) + \sum_{k=1}^{K} x^k - \sum_{k=1}^{K} y^k .$$

If both x and y correspond to probability distributions i.e. $\sum_{k=1}^{K} x^k = \sum_{k=1}^{K} y^k = 1$, then the induced Bregman divergence corresponds to the standard KL-divergence between the two distributions. For multi-class classification, $\mathcal{Z} \subseteq \mathbb{R}^{p \times K}$ and each $z^i \in \Delta_K$ where Δ_K is K-dimensional simplex. We will refer to coordinate j of z^i as $[z^i]_j$. Since $z^i \in \Delta_K$, $[z^i]_k \geq 0$ and $\sum_{k=1}^{K} [z^i]_k = 1$.

Let us instantiate the general SMD updates in Eq. (5) when using the negative entropy mirror map. In this case, for $z \in \Delta_K$, $[\nabla \phi(z)]_k = 1 + \log([z]_k)$. Denoting $\nabla_t := \nabla_z \ell_t(z_t)$ and using $[\nabla_t]_k$ to refer to coordinate k of the K-dimensional vector ∇_t. Hence, Eq. (5) can be written as:

$$[z^{i_t}_{t+1/2}]_k = [{z_t}^{i_t}]_k \, \exp\left(-\eta_t [\nabla_t]_k\right) \tag{7}$$

For multi-class classification, $y^i \in \{0, 1\}^K$ are one-hot vectors. If $[y^i]_k$ refers to coordinate k of vector y^i, then corresponding log-likelihood for n observations can be written as:

$$\ell(z) = \sum_{i=1}^{n} \sum_{k=1}^{K} [y^i]_k \log([z^i]_k) .$$

In our target optimization framework, the targets correspond to the probabilities of classifying the points into one of the classes. We use a parameterization to model the vector-valued function $f_i(\theta) : \mathbb{R}^d \to \mathbb{R}^K$. This ensures that for all i, $\sum_{k=1}^{K} [f_i(\theta)]_k = 1$. Hence, the projection step in Eq. (5) can be rewritten as:

$$\min_{z \in \mathcal{Z}} D_\phi(z, z_{t+1/2}) = \sum_{k=1}^{K} [f_{i_t}(\theta)]_k \log\left(\frac{[f_{i_t}(\theta)]_k}{[z^{i_t}_{t+1/2}]_k}\right)$$

where $[z^{i_t}_{t+1/2}]_k$ is computed according to Eq. (7). Since the computation of $[z^{i_t}_{t+1/2}]_k$ and the resulting projection only depends on sample i_t, it can be implemented efficiently.

Appendix C. Proofs

Lemma 1 *Assuming that $g_t^z(\theta)$ is β-smooth w.r.t. the Euclidean norm and $\eta \leq \frac{1}{L}$, then, for $\alpha = 1/\beta$, iteration t of Algorithm 1 guarantees that $h(\theta_{t+1}) \geq h(\theta_t)$ for any number $m \geq 1$ of surrogate steps. In this setting, under the additional assumption that h is lower-bounded by h^*, then Algorithm 1 results in the following guarantee,*

$$\min_{t \in \{0,\dots,T-1\}} \|\nabla h(\theta_t)\|_2^2 \leq \frac{2\beta\,[h(\theta_0) - h^*]}{T}.$$

Proof Using the update in Algorithm 1 with $\alpha = \frac{1}{\beta}$ and the β-smoothness of $g_t^z(\theta)$, for all $k \in [m-1]$,

$$g_t^z(\omega_{k+1}) \leq g_t^z(\omega_k) - \frac{1}{2\beta} \|\nabla g_t^z(\omega_k)\|_2^2$$

After m steps,

$$g_t^z(\omega_m) \leq g_t^z(\omega_0) - \frac{1}{2\beta} \sum_{k=0}^{m-1} \|\nabla g_t^z(\omega_k)\|_2^2$$

Since $\theta_{t+1} = \omega_m$ and $\omega_0 = \theta_t$ in Algorithm 1,

$$\implies g_t^z(\theta_{t+1}) \leq g_t^z(\theta_t) - \frac{1}{2\beta} \|\nabla g_t^z(\theta_t)\|_2^2 - \sum_{k=1}^{m-1} \|\nabla g_t^z(\omega_k)\|_2^2$$

Note that $h(\theta_t) = g_t^z(\theta_t)$ and if $\eta \leq \frac{1}{L}$, then $h(\theta_{t+1}) \leq g_t^z(\theta_{t+1})$. Using these relations,

$$h(\theta_{t+1}) \leq h(\theta_t) - \left[\underbrace{\frac{1}{2\beta} \|\nabla g_t^z(\theta_t)\|_2^2 + \sum_{k=1}^{m-1} \|\nabla g_t^z(\omega_k)\|_2^2}_{\text{positive}}\right] \implies h(\theta_{t+1}) \leq h(\theta_t).$$

This proves the first part of the Lemma. Since $\sum_{k=1}^{m-1} \|\nabla g_t^z(\omega_k)\|_2^2 \geq 0$,

$$h(\theta_{t+1}) \leq h(\theta_t) - \frac{1}{2\beta} \|\nabla g_t^z(\theta_t)\|_2^2 \implies \|\nabla h(\theta_t)\|_2^2 \leq 2\beta\,[h(\theta_t) - h(\theta_{t+1})]$$

(Since $\nabla h(\theta_t) = \nabla g_t^z(\theta_t)$)

Summing from $k = 0$ to $T-1$, and dividing by T,

$$\frac{\|\nabla h(\theta_t)\|_2^2}{T} \leq \frac{2\beta\,[h(\theta_t) - h^*]}{T} \implies \min_{t \in \{0,\dots,T-1\}} \|\nabla h(\theta_t)\|_2^2 \leq \frac{2\beta\,[h(\theta_t) - h^*]}{T}$$

■

C.1. Equivalence of Optimizing Surrogate and SGD in Target Space

Optimization in target space is

$$\begin{aligned} z_{t+1/2} &= z_t - \eta_t \nabla_z \ell_t(z_t) \\ \bar{z}_{t+1} &= \arg\min_{z \in \mathcal{Z}} \frac{1}{2} \left\| z_{t+1/2} - z \right\|_{\mathcal{P}_t}^2 . \end{aligned} \tag{8}$$

where, $\mathcal{P}_t \in \mathbb{R}^{p \times p}$ is a diagonal preconditioner such that $\mathcal{P}_{i_t,i_t} = 1$ and $\mathcal{P}_{i,j} = 0$ for all $i, j \neq i_t$. Since ℓ is separable, if we assume i_t is a coordinate sampled from z, we can rewrite the $z_{t+1/2}$ update as follows:

$$\begin{aligned} {z_{t+1/2}}^{i_t} &= {z_t}^{i_t} - \eta_t \nabla_z \ell_{i_t}(z_t) \\ {z_{t+1/2}}^{j} &= {z_t}^{j} \quad \text{when } j \neq i_t . \end{aligned}$$

Putting the above update in the projection step we have

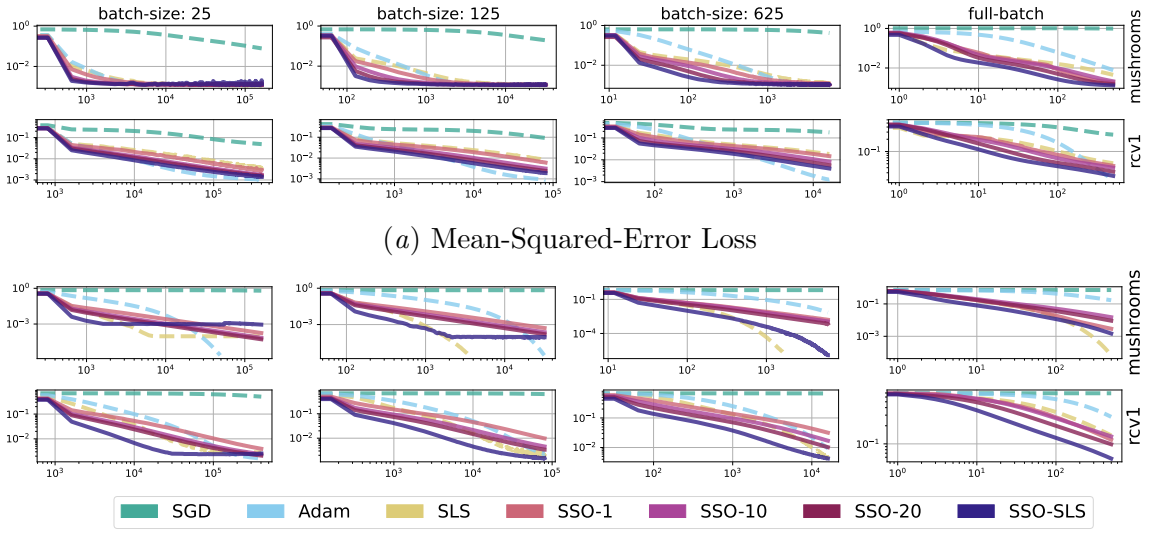
$$\begin{aligned} \bar{z}_{t+1} &= \arg\min_{z \in \mathcal{Z}} \frac{1}{2} \left\| z_{t+1/2} - z \right\|_{\mathcal{P}_t}^2 \\ &= \arg\min_{z \in \mathcal{Z}} \frac{1}{2} \left\{ 0 \times \sum_{j=1, j \neq i_t} \left\| {z_{t+1/2}}^{j} - z^j \right\|_2^2 + 1 \times \left\| {z_{t+1/2}}^{i_t} - z^{i_t} \right\|_2^2 \right\} = \arg\min_{z \in \mathcal{Z}} \{ \left\| {z_{t+1/2}}^{i_t} - z^{i_t} \right\|_2^2 \} \\ &= \arg\min_{z \in \mathcal{Z}} \frac{1}{2} \{ \left\| {z_t}^{i_t} - \eta_t \nabla_z \ell_{i_t}(z_t) - z^{i_t} \right\|_2^2 \} = \arg\min_{z \in \mathcal{Z}} \left\{ \frac{\partial \ell_{i_t}(z_t)}{\partial z^{i_t}} [z^{i_t} - z_t^{i_t}] + \frac{1}{2\eta_t} \left\| z^{i_t} - z_t^{i_t} \right\|_2^2 \right\} \\ & \qquad \text{(Due to separability of } \ell) \end{aligned}$$

Since for all $z \in \mathcal{Z}$, $z = f(\theta)$ and $z^i = f_i(\theta)$ for all i. Hence $\bar{z}_{t+1} = f(\bar{\theta}_{t+1})$ such that,

$$\bar{\theta}_{t+1} = \arg\min_{\theta \in \Theta} \left\{ \frac{\partial \ell_{i_t}(z_t)}{\partial z^{i_t}} [f_{i_t}(\theta) - f_{i_t}(\theta_t)] + \frac{1}{2\eta_t} \| f_{i_t}(\theta) - f_{i_t}(\theta_t) \|_2^2 \right\} = \arg\min_{\theta \in \Theta} \tilde{g}_t^z(\theta)$$

Since $\bar{z}_{t+1} = f(\bar{\theta}_{t+1})$, for all j, $z_{t+1}^j = f_j(\bar{\theta}_{t+1})$, and hence all coordinates of $\bar{z}_{t+1}$ have been updated (though not necessarily via a stochastic gradient update).

Appendix D. Additional Experimental Results



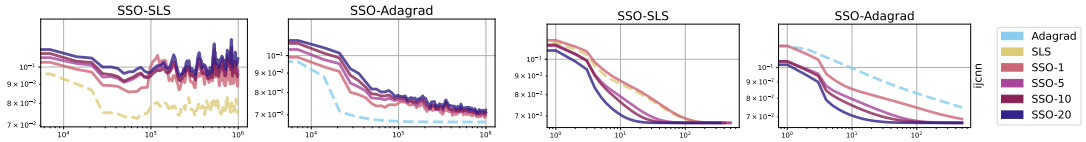
(a) Mean-Squared-Error Loss

(b) Logistic Loss

Figure 2: Comparison of average squared loss and logistic loss between `SGD`, `SLS`, `Adam`, and `SSO-SLS`. The number next to `SSO` in the legend indicates the number of steps taken on the surrogate. All plots are in log-log space. We note the `SGD` with its theoretical step-size is outperformed by more sophisticated algorithms like `SLS` or `Adam`. In contrast, `SSO` with the theoretical step-size is competitive with both `SLS` and `Adam` with default hyper-parameters.

Supervised Learning: We evaluate our framework on the LibSVM benchmarks [3], a standard suite of convex-optimization problems. Here consider two datasets – `mushrooms`, and `rcv1`, two losses – squared loss and logistic loss, and four batch sizes – {25, 125, 625, full-batch} for the linear parameterization ($f = X^\top \theta$). Each optimization algorithm is run for 500 epochs (full passes over the data). We compare stochastic surrogate optimization (`SSO`), against `SGD` with the standard theoretical $1/2L_\theta$ step-size, `SGD` with the step-size set according to a stochastic line-search [24] `SLS`, and finally `Adam` [10] using default hyper-parameters. Since `SSO` is equivalent to projected SGD in the target space, we set η (in the surrogate definition) to $1/2L$ where L is the smoothness of ℓ w.r.t z. For squared loss, L is therefore set to 1, while for logistic it is set to 2. Fig. 2(a) and Fig. 2(b) show that (i) `SSO` improves over `SGD` when the step-sizes are set theoretically, (ii) `SSO` is competitive with `SLS` or `Adam`, and (iii) as m increases, on average, the performance of `SSO` improves as projection error decreases.

Extensions to the Stochastic Surrogate: Next we consider using two other standard optimization algorithms (`SLS` and `Adagrad`) in the target space, to construct new surrogates. We refer to the resulting algorithms as `SSO-SLS` and `SSO-AdaGrad`. For `SSO-SLS`, at every stage, we perform a backtracking Armijo line-search in target space. This means that for every stochastic gradient $\nabla_z \ell_{i_t}(z)$, we find an step-size η_t which satisfies the Armijo condition



(*a*) Stochastic Setting (batch-size=5) (*b*) Deterministic Setting

Figure 3: Comparison of two variations of the `SSO` algorithm on the `ijcnn` dataset in deterministic and stochastic settings. We compare against parametric `SLS`, and `Adagrad`. The left plot considers target-space variation of `SLS` to its parametric counterpart, while the plot on the right compares the `Adagrad` variation. Notice (i) each SSO variation improves over its parametric counter-part in the deterministic setting with significant improvements for large m, and (ii) in the stochastic setting each SSO variation inherits the properties of its parent optimizer.

in the target space: $\ell_{i_t}(z_t - \eta_t \nabla_z \ell_{i_t}(z_t)) \leq \ell_{i_t}(z_t) - \frac{\eta_t}{2} \|\nabla_z \ell_{i_t}(z_t)\|_2^2$. We then follow the same procedure as before, taking steps according to this η_t using Algorithm 1. Similarly, following the original Adagrad algorithm [5], we set η_t according to $1/\sqrt{\sum_{i=1}^{t} ||\nabla_z \ell_i(z_i)||^2}$. A comparison of each algorithm along-side its parametric variant is included in Fig. 3, which shows (i) in fully deterministic settings, SSO can lead to large improvements over its parametric counterparts, and (ii) in the stochastic setting, SSO inherits the characteristics of the optimization algorithm used to derive its surrogate (for SLS we converge to a neighbourhood of the minimizer, while for Adagrad we observe a monotonic decrease in the loss). We also include `SSO-SLS` in Fig. 2(*a*) and Fig. 2(*b*) with $m = 20$ and observe that it results in a significant and consistent improvement over standard baselines.

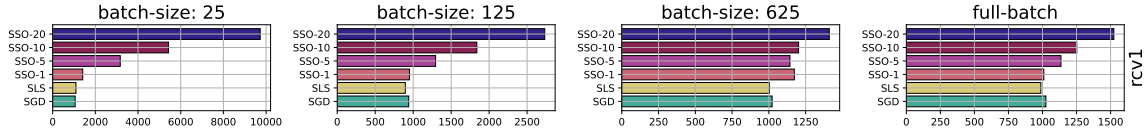


Figure 4: Comparison of run-times (in seconds) between SSO and relevant baselines. These plots illustrate that when moving data onto GPU is costly (as in the full-batch setting) SSO will be roughly as fast as even SGD.

Runtime Comparison: These experiments demonstrate the relative runtime between algorithms for the supervised learning setting. Each column provides run-times evaluated over 500 epochs for batch-sizes of 25, 125, 625 and full-batch respectively. Evaluation is performed on the `rcv1` dataset [3]. This dataset was chosen as it is large enough to incur significant wall-clock time to move data to/from the GPU. All experiments were run using the same resources, namely a NVIDIA GeForce RTX 2070 graphics card with a AMD Ryzen 9 3900 12-Core Processor. Fig. 4 shows that for small batch-sizes, the time it takes to move data to/from the GPU is small and the target variants can be slower than their parametric counterparts. However, for large-batch sizes, that require longer time to be moved into memory, we observe that the time for the additional surrogate steps in `SSO`

is largely amortized. This experiment shows that in cases where data-access is the major bottleneck in computing the stochastic gradient, target optimization can be beneficial.

D.1. Stochastic Surrogate Optimization

Comparisons of `SGD`, `SLS`, `Adam`, `Adagrad`, and `SSO` evaluated on three SVMLib benchmarks `mushrooms`, `ijcnn`, and `rcv1`. Each run was evaluated over three random seeds following the same initialization scheme. All plots are in log-log space to make trends between optimization algorithms more apparent. As before in all settings, algorithms use either their theoretical step-size when available, or the default as defined by [17]. The inner-optimization loop are set according to line-search parameters and heuristics following Vaswani et al. [24]. All algorithms and batch sizes are evaluated for 500 epochs and performance is represented as a function of total optimization steps. Below we include three different step-size schedules: constant, $\frac{1}{\sqrt{t}}$ [16], and $(1/T)^{t/T}$ [26]. For further details see the attached code repository.

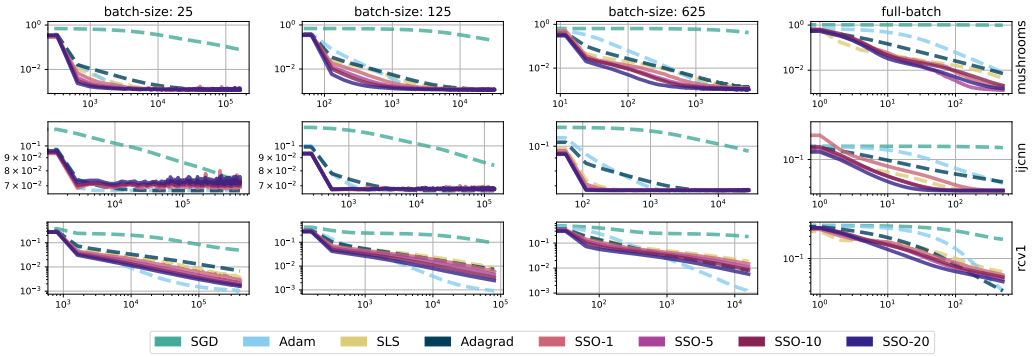


Figure 5: Constant step-size: comparison of optimization algorithms under a **mean squared error loss**. We note here, as in Fig. 2(a), `SSO` significantly outperforms its parametric counterpart, and maintains performance which is on par with both `SLS` and `Adam`. Additionally we note that taking additional steps in the surrogate generally improves performance.

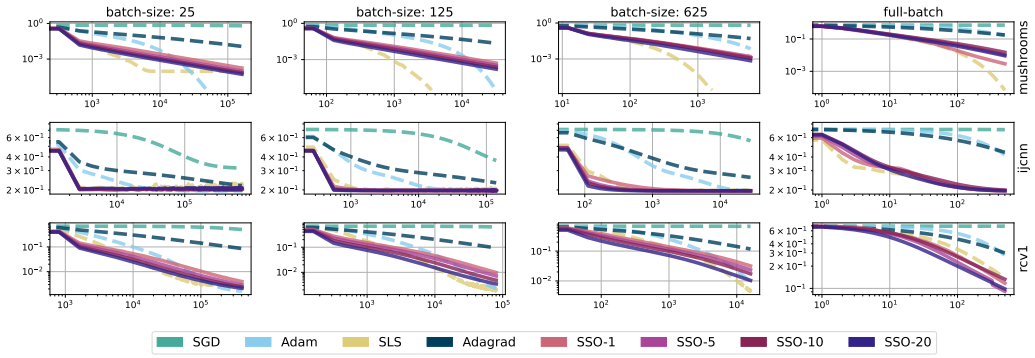


Figure 6: Constant step-size: comparison of optimization algorithms under a average **logistic loss**. We note here, as in Fig. 2(b), `SSO` significantly outperforms its parametric counterpart, and maintains performance which is on par with both `SLS` and `Adam`. Additionally we note that taking additional steps in the surrogate generally improves performance

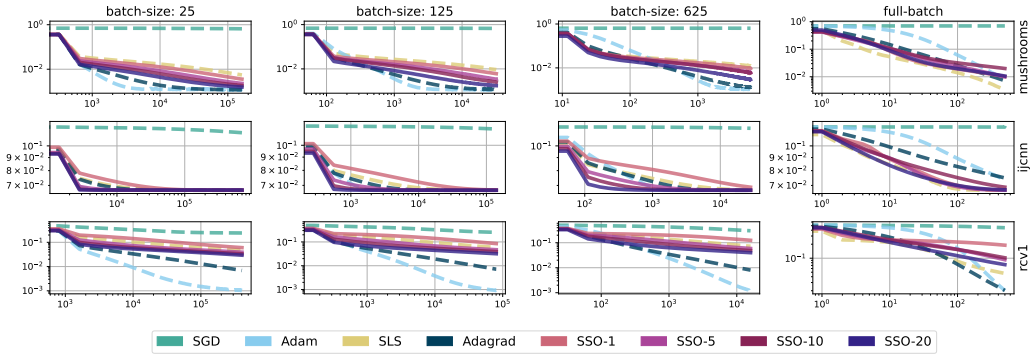


Figure 7: Decreasing step-size: comparison of optimization algorithms under a **mean squared error loss**. In this figure, unlike Fig. 2(a), we compare examples which include a decaying step size of $\frac{1}{\sqrt{t}}$ alongside both `SSO` as well as `SGD` and `SLS`. `Adam` (and `Adagrad`) remain the same as in Fig. 2(a). Again, we note that taking additional steps in the surrogate generally improves performance. Additionally the decreasing step-size seems to help maintain strict monotonic improvement.

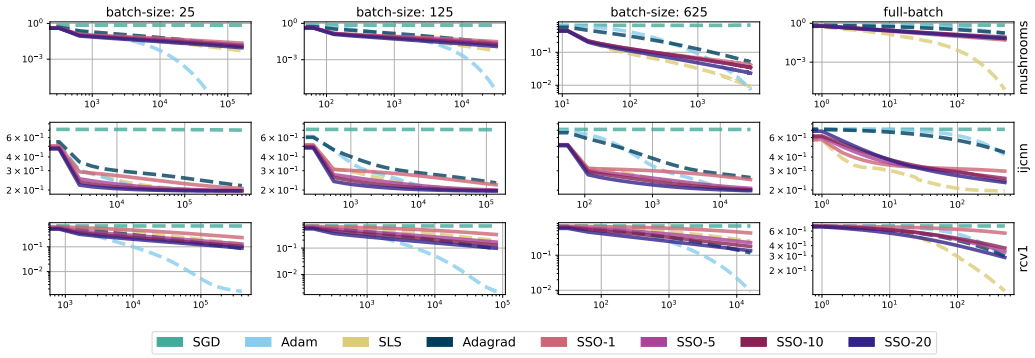


Figure 8: Decreasing step-size: comparison of optimization algorithms under a **logistic loss**. In this figure, unlike Fig. 2(*b*), we compare examples which include a decaying step size of $\frac{1}{\sqrt{t}}$ alongside both `SSO` as well as `SGD` and `SLS`. `Adam` (and `Adagrad`) remain the same as in Fig. 2(*b*). Again, we note that taking additional steps in the surrogate generally improves performance. Additionally the decreasing step-size seems to help maintain strict monotonic improvement.

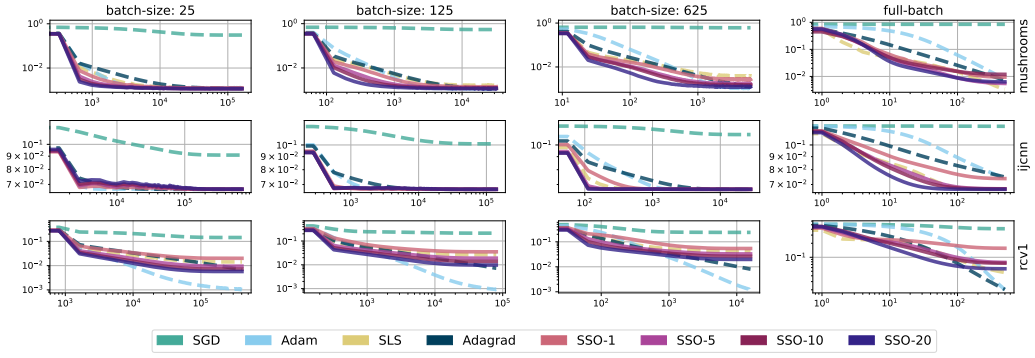


Figure 9: Exponential step-size: comparison of optimization algorithms under a **mean squared error loss**. In this figure, unlike Fig. 2(*b*), we compare examples which include a decaying step size of $(\frac{1}{T})^{t/T}$ alongside both `SSO` as well as `SGD` and `SLS`. `Adam` remains the same as in Fig. 2(*b*). Again, we note that taking additional steps in the surrogate generally improves performance. Additionally the decreasing step-size seems to help maintain strict monotonic improvement. Lastly, because of a less aggressive step size decay, the optimization algorithms make more progress then their stochastic $\frac{1}{\sqrt{t}}$ counterparts.

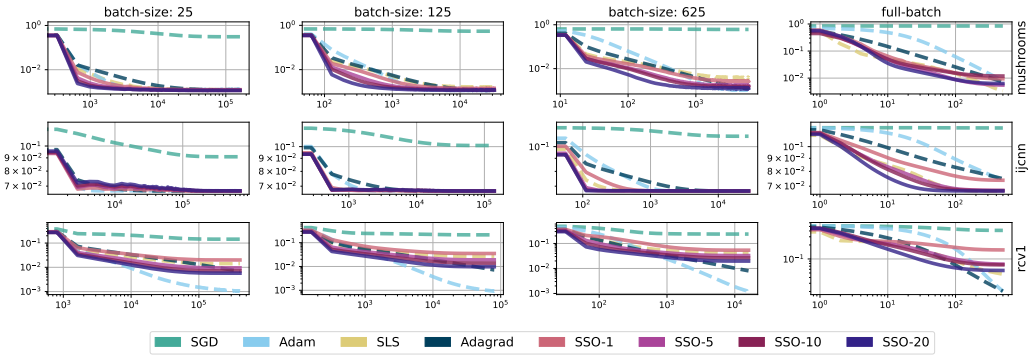


Figure 10: Exponential step-size: comparison of optimization algorithms under a **logistic loss**. In this figure, unlike Fig. 2(*b*), we compare examples which include a decaying step size of $(\frac{1}{T})^{t/T}$ alongside both `SSO` as well as `SGD` and `SLS`. `Adam` remains the same as in Fig. 2(*b*). Again, we note that taking additional steps in the surrogate generally improves performance. Additionally the decreasing step-size seems to help maintain strict monotonic improvement. Lastly, because of a less aggressive step size decay, the optimization algorithms make more progress then their stochastic $\frac{1}{\sqrt{t}}$ counterparts.

D.2. Stochastic Surrogate Optimization with a Line-search

Comparisons of `SGD`, `SLS`, `Adam`, `Adagrad`, and `SSO-SLS` evaluated on three SVMLib benchmarks `mushrooms`, `ijcnn`, and `rcv1`. Each run was evaluated over three random seeds following the same initialization scheme. All plots are in log-log space to make trends between optimization algorithms more apparent. As before in all settings, algorithms use either their theoretical step-size when available, or the default as defined by [17]. The inner-optimization loop are set according to line-search parameters and heuristics following Vaswani et al. [24]. All algorithms and batch sizes are evaluated for 500 epochs and performance is represented as a function of total optimization steps.

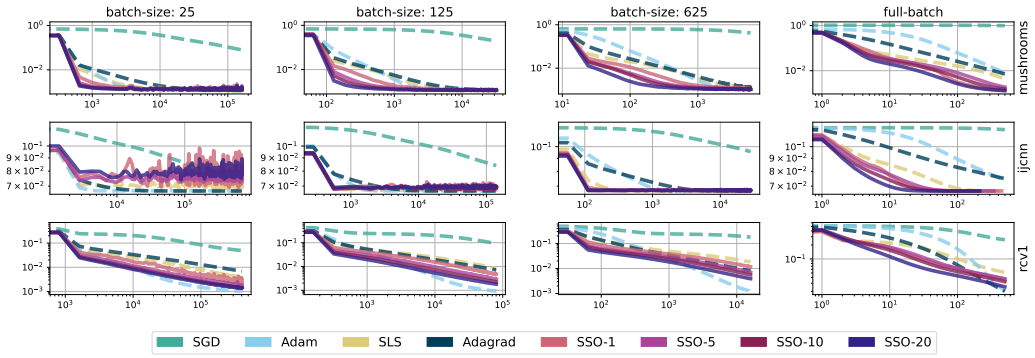


Figure 11: Constant step-size: comparison of optimization algorithms under a **mean squared error loss**. We note here, as in Fig. 2(*a*), `SSO-SLS` outperforms its parametric counterpart, and maintains performance which is on par with both `SLS` and `Adam`. Additionally we note that taking additional steps in the surrogate generally improves performance, especially in settings with less noise (full-batch and batch-size 625).

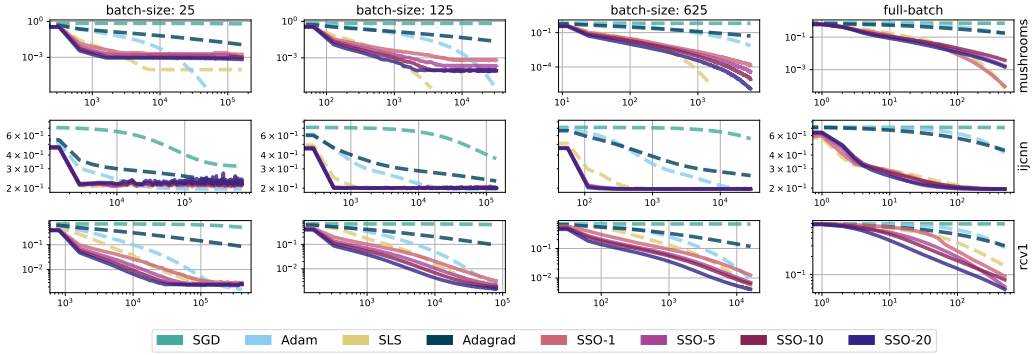


Figure 12: Constant step-size: comparison of optimization algorithms under a average **logistic loss**. We note here, as in Fig. 2(*b*), `SSO-SLS` outperforms its parametric counterpart, and maintains performance which is on par with both `SLS` and `Adam`. Additionally we note that taking additional steps in the surrogate generally improves performance.

D.3. Combining Stochastic Surrogate Optimization with Adaptive Gradient Methods

Comparisons of `SGD`, `SLS`, `Adam`, `Adagrad`, and `SSO-Adagrad` evaluated on three SVMLib benchmarks `mushrooms`, `ijcnn`, and `rcv1`. Each run was evaluated over three random seeds following the same initialization scheme. All plots are in log-log space to make trends between optimization algorithms more apparent. As before in all settings, algorithms use either their theoretical step-size when available, or the default as defined by [17]. The inner-optimization loop are set according to line-search parameters and heuristics following Vaswani et al. [24]. All algorithms and batch sizes are evaluated for 500 epochs and performance is represented as a function of total optimization steps. Here unlike in Appendix D.1, we update the η according to the same schedule as scalar `Adagrad` (termed AdaGrad-Norm in Ward et al.

[27]), as discussed in Section 5. Because Adagrad does not have an easy to compute optimal theoretical step size, for our setting we set the log learning rate (the negative of log η) to be 2.. For further details see the attached coding repository.

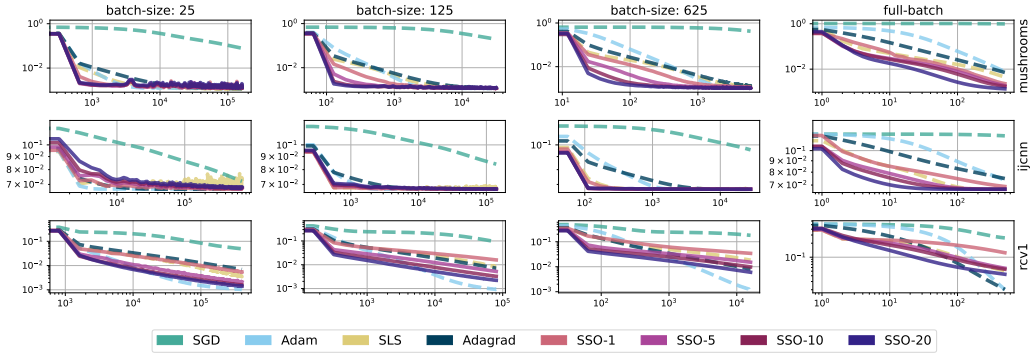


Figure 13: Comparison in terms of average MSE loss of `SGD`, `SLS`, `Adam`, and `SSO-Adagrad` evaluated under a **mean squared error loss**. These plots show that `SSO-Adagrad` outperforms its parametric counterpart, and maintains performance which is on par with both `SLS` and `Adam`. Additionally, we again find that taking additional steps in the surrogate generally improves performance.

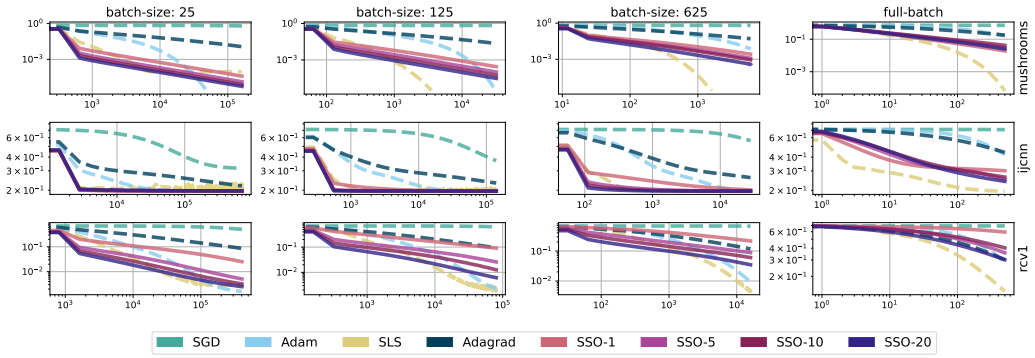


Figure 14: Comparison in terms of average MSE loss of `SGD`, `SLS`, `Adam`, and `SSO-Adagrad` evaluated under a **logistic loss**. These plots show that `SSO-Adagrad` outperforms its parametric counterpart, and maintains performance which is on par with both `SLS` and `Adam`. Additionally, we again find that taking additional steps in the surrogate generally improves performance.

D.4. Combining Stochastic Surrogate Optimization With Online Newton Steps

Comparisons of `SGD`, `SLS`, `Adam`, `Adagrad`, and `SSO-Newton` evaluated on three SVMLib benchmarks `mushrooms`, `ijcnn`, and `rcv1`. Each run was evaluated over three random seeds following the same initialization scheme. All plots are in log-log space to make trends between optimization algorithms more apparent. As before, in all settings, algorithms use either their theoretical step-size when available, or the default as defined by [17]. The inner-optimization loop are set according to line-search parameters and heuristics following Vaswani et al. [24]. All algorithms and batch sizes are evaluated for 500 epochs and performance is represented as a function of total optimization steps. Here unlike in Appendix D.1, we update the η according to the same schedule as `Online Newton`. We omit the MSE example as `SSO-Newton` in this setting is equivalent to `SSO`. In the logistic loss setting however, which is displayed below, we re-scale the regularization term by $(1-p)p$ where $p = \sigma(f(x))$ where σ is this sigmoid function, and f is the target space. This operation is done per-data point, and as can be seen below, often leads to extremely good performance, even in the stochastic setting. In the plots below, if the line vanishes before the maximum number of optimization steps have occurred, this indicates that the algorithm has converged to the minimum and is no longer executed. Notably, `SSO-Newton` achieves this in for multiple data-sets and batch sizes.

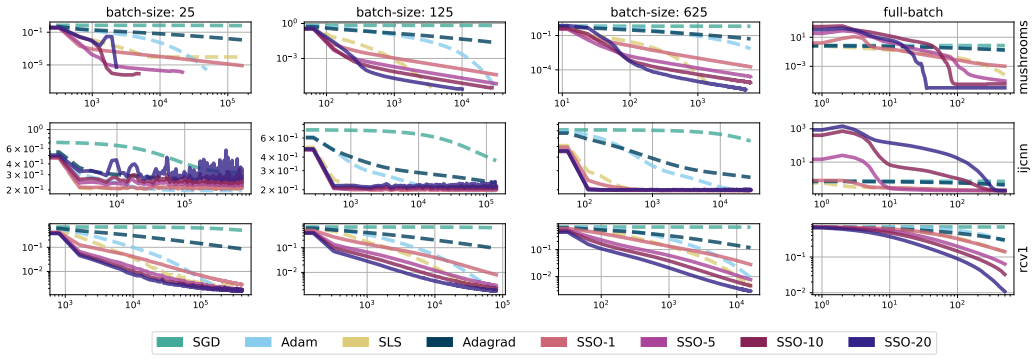


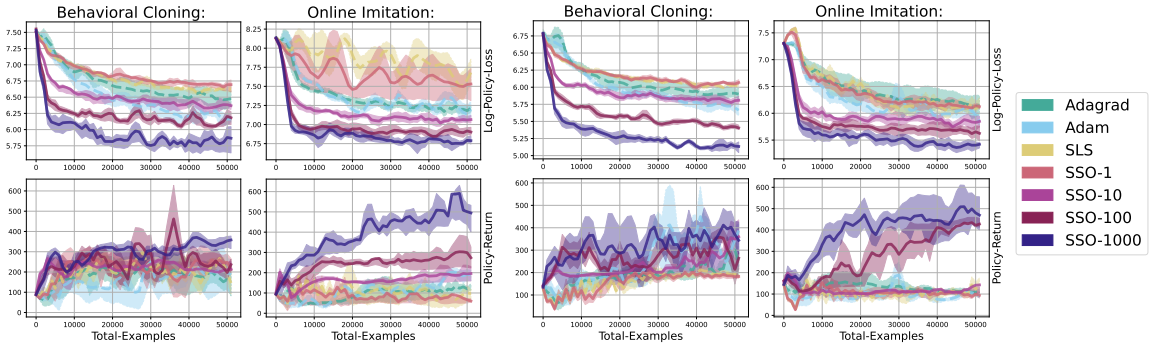
Figure 15: Comparison in terms of average logistic loss of `SGD`, `SLS`, `Adam`, and `SSO-Newton` evaluated on the **logistic loss**. This plot displays that significant improvement can be made at no additional cost by re-scaling the regularization term correctly. Note that in the case of mushrooms, `SSO-Newton` in all cases for $m = 20$ reaches the stopping criteria before the 500th epoch. Second, even in many stochastic settings, `SSO-Newton` outperforms both `SLS` and `Adam`.

D.5. Imitation Learning

Comparisons of `Adagrad`, `SLS`, `Adam`, and `SSO` evaluated on two Mujoco [23] imitation learning benchmarks [11], `Hopper-v2`, and `Walker-v2`. In this setting training and evaluation proceed in rounds. At every round, a behavioral policy samples data from the environment, and an expert labels that data. The goal is guess at the next stage (conditioned on the sampled states) what the expert will label the examples which are gathered. Here, unlike the supervised learning setting, we receive a stream of new data points which can be correlated and drawn from following different distributions through time. Theoretically this makes the optimization problem significantly more difficult, and because we must interact with a simulator, querying the stochastic gradient can be expensive. Like the example in Appendix D, in this setting we will interact under both the experts policy distribution (behavioral cloning), as well as the policy distribution induced by the agent (online imitation). Like Appendix D, we will again parameterize a standard normal distribution whose mean is learned through a mean squared error loss between the expert labels and the mean of the agent policy. Again, the expert is trained following soft-actor-critic.

In this this setting we evaluate two measures: the per-round log-policy loss, and the policy return. The log policy loss is as described above, while the return is a measure of how well the imitation learning policy actually solves the task. In the Mujoco benchmarks, this reward is defined as a function of how quickly the agent can move in space, as well as the power exerted to move. Imitation learning generally functions by taking a policy which has a high reward (e.g. can move through space with very little effort in terms of torque), and directly imitating it instead of attempting to learn a cost to go function as is done in RL [22].

Each algorithm is evaluated over three random seeds following the same initialization scheme. As before, in all settings, algorithms use either their theoretical step-size when available, or the default as defined by [17]. The inner-optimization loop is set according to line-search parameters and heuristics following Vaswani et al. [24]. All algorithms and batch sizes are evaluated for 50 rounds of interaction and performance is represented as a function of total interactions with the environment. Unlike the imitation learning experiments in Section 5, here we learn a policy which is parameterized by a two layer perceptron with 256 hidden units and relu activations. For further details, please see the attached code repository.



(*a*) Walker2d-v2 Environment (*b*) Hopper-v2 Environment

Figure 16: Comparison of policy return, and log policy loss incurred by `SGD`, `SLS`, `Adam`, `Adagrad`, and `SSO` as a function of the total interactions. Unlike Section 5, the mean of the policy is parameterized by a **neural network model**. In both environments, for both behavioral policies, `SSO` outperforms all other online-optimization algorithms. Additionally, as m in increases, so to does the performance of `SSO` in terms of both the return as well as the loss.