JOINT DESCENT: TRAINING AND TUNING SIMULTANEOUSLY

Anonymous authors
Paper under double-blind review

ABSTRACT

Typically in machine learning, training and tuning are done in an alternating manner: for a fixed set of hyperparameters y, we apply gradient descent to our objective $f(x, y)$ over trainable variables x until convergence; then, we apply a tuning step over y to find another promising setting of hyperparameters. Because the full training cycle is completed before a tuning step is applied, the optimization procedure greatly emphasizes the gradient step, which seems justified as first-order methods provides a faster convergence rate. In this paper, we argue that an equal emphasis on training and tuning lead to faster convergence both theoretically and empirically. We present Joint Descent (JD) and a novel theoretical analysis of acceleration via an unbiased gradient estimate to give an optimal iteration complexity of $O(\sqrt{\kappa}n_y \log(n/\epsilon))$, where κ is the condition number and n_y is the dimension of y. This provably improves upon the naive classical bound and implies that we essentially train for free if we apply equal emphasis on training and tuning steps. Empirically, we observe that an unbiased gradient estimate achieves the best convergence results, supporting our theory.

1 INTRODUCTION

For many practical machine learning tasks, the importance of efficient hyperparameter tuning in order to achieve state-of-the-art performance is often overlooked. Because modern neural network architectures and learning models have become increasingly complicated, the need to precisely and efficiently tune the growing number of hyperparameters has also increased in traditional supervised learning (Snoek et al., 2012), as well as diverse applications such as reinforcement learning (Choromanski et al., 2018), neural network attacks (Papernot et al., 2017), and generative adversarial networks (Goodfellow et al., 2014). Traditionally, hyperparameter tuning has been done in the alternating regime where the learner alternates between choosing the hyperparameters and optimizing the model with that set of hyperparameters. The main difficulty that arises is the computational cost of such an approach, which incurs the full computational cost of model training for every attempted configuration of hyperparameters. Parallelism has alleviated some of the computational burden (Falkner et al., 2018), but does not change the total computational cost and has its own drawbacks since sequential optimization will in general provide the best solutions.

Even in academic literature, the understanding of optimization algorithms and convergence has been divided into two camps: first-order vs zeroth-order optimization. Classical first-order optimization assumes that the gradient of the objective function can be efficiently computed and the intuitive idea of applying vanilla gradient descent gives a $O(\kappa \log(R^2/\epsilon))$ iteration complexity for converging to within ϵ of the optimum of an L-Lipschitz convex function with condition number κ and $R = \|x_0 - x^*\|$ (Nesterov, 2013). Acceleration, in the form of momentum, was then applied successfully to achieve a $O(\sqrt{\kappa} \log(R^2/\epsilon))$ iteration complexity and this convergence bound is optimal. Furthermore, many variants of gradient descent have arose in different settings, such as stochastic gradient descent (SGD) (Bottou, 2010), mirror descent (Beck & Teboulle, 2003), coordinate gradient descent (Tseng & Yun, 2009), or proximal gradient descent (Nitanda, 2014).

Zeroth-order optimization, on the other hand, assumes that the learner only has access to function evaluations as opposed to gradient evaluations. Not surprisingly, the classical approach to zeroth-order optimization is to simply estimate the gradients from function values and essentially reduce to first-order optimization (Flaxman et al., 2005). Nesterov & Spokoiny (2011) analyze this class of algorithms as gradient descent on a Gaussian smoothing of the objective and gives an accelerated $O(n\sqrt{\kappa}\log(LR^2/\epsilon))$ iteration complexity where n is the input dimension. This approach was extended by (Duchi et al., 2015) for stochastic settings, by (Ghadimi & Lan, 2013) for nonconvex settings, and by (Shamir, 2017) for non-smooth and general ℓ_p norm settings. Also, first-order techniques such as variance reduction (Liu et al., 2018b), conditional gradients (Balasubramanian & Ghadimi, 2018), and diagonal preconditioning (Mania et al., 2018) have been successfully transfered to the zeroth-order setting.

However, estimating gradients can lead to bad optima when optimizing non-convex, noisy, or discrete functions. To go beyond simply mimicking first-order techniques, researchers have proposed two alternate directions of tackling zeroth order optimization. The first approach is a local randomized descent based approach that relies only on the ranking of function evaluations. These approaches tend to be more robust when gradients suffer from high variance due to non-robust local minima or highly non-smooth objectives, which are common in the fields of deep learning and reinforcement learning (Mania et al., 2018). Examples include direct search via deterministic positive spanning sets (Brooks, 1958), evolutionary or genetic strategies such as CMA-ES (Auger & Hansen, 2005), and stochastic gradientless descent (Golovin et al., 2019; Gorbunov et al., 2019). While these methods usually have worse theoretical bounds of $O(n\kappa\log(LR^2/\epsilon))$ with a linear dependence on κ, it is noted that these methods are invariant under monotone transforms and affine projections of the objective, a property that might explain their robustness in practice, especially in high-variance, high-dimensional settings (Golovin et al., 2019). Furthermore, these rank-based approaches do not suffer from the numerical instability of approximating gradients, both in practice and in the analysis.

The second approach is a model based approach that first builds a model of the objective from the data, and then optimizes an acquisition that is a function of the outputs of the model. The model is often probabilistic and the setting is thus referred to as as Bayesian optimization. In general, these methods attempt to solve a broader more global non-convex optimization with multiple optima and usually have have weaker theoretical regret bounds (Frazier, 2018). Furthermore, the computational cost is significant; for example, Bayesian optimization generally has theoretical regret bounds that grow exponentially in dimension and each iteration assumes the success of an inner optimization loop of the acquisition function (Srinivas et al., 2009). However, they often do well empirically on a variety of blackbox optimization problems and have become a staple in zeroth-order optimization in the recent years (Hansen et al., 2009).

Because of the relative division of first-order and zeroth-order optimization in literature, as well as in practical software implementation, training and tuning has been traditionally done separately in an alternating, thereby computationally expensive, manner. Thus, there has been a recent push to speed up optimization via joint training and tuning, such as automatic updating of learning rate and momentum for SGD during training (Lancewicki & Kopru, 2019), tuning of both hyperparameter and parameters for SVMs (Liao & Jia, 2007), or self-tuning networks that simultaneously learn the network weights as well as the best hyperparameter response function (MacKay et al., 2019), activation function (Hou et al., 2017), or architecture (Cortes et al., 2017; Liu et al., 2018a). Furthermore, a technique to compute hyperparameter gradient during the course of training was proposed in (Maclaurin et al., 2015) and there are gradient-based approaches to simultaneously train and tune optimization hyperparameters (Andrychowicz et al., 2016) and regularization hyperparameters (Luketina et al., 2016). Lastly, Population-based Training (PBT) was recently proposed to jointly train variables via gradients and hyperparameters via evolutionary strategies (Jaderberg et al., 2017). Although these papers provide a good overview on the empirical benefits of applying joint optimization, especially in computational savings, across a diverse array of settings, they provide little theoretical understanding or justifications.

1.1 Our setup

In this paper, we consider the problem of jointly minimizing a bivariate function of the form $f(x, y)$, where x denotes the trainable variables and y are the hyperparameters. Specifically, let n_x, n_y denote the dimension of x, y respectively and let $n = n_x + n_y$. Our goal is to solve

$$\min_{x,y} f(x, y) \tag{1}$$

with as few iterations as possible, where each iteration allows either:

1. Evaluation of a partial derivative with respect to x: $\nabla_x f(x, y)$
2. Evaluation of function value at (x, y): $f(x, y)$

It may seem inpractical that an zeroth-order evaluation costs the same as a first-order derivative evaluation, but this occurs in many settings where the derivatives are cheap to evaluate, most notable in backpropagation for neural networks (Hecht-Nielsen, 1992), where the gradient evaluation computation cost is only a constant factor more expensive than that of evaluating the network. Furthermore, modern auto-differentiation software and hardware often optimize for the speed of gradient computations, which has become extremely fast and cheap.

However, it is equally important to note that there are instances in machine learning where the gradient evaluation is significantly more expensive than a function evaluation. For example, in meta-learning or multi-task learning, objective functions are often of the form $f(x - \eta\nabla f(x))$, which requires a full Hessian vector product if a full gradient is computed (Finn et al., 2017). By converting some of the expensive gradient computation into cheap function evaluation computation, one can benefit from joint optimization in this setting to improve runtime.

Classically, problem 1 is solved by first minimizing over variables x and reducing to a zeroth order problem over hyperparameters y on the optimized function $g(y) = \min_x f(x, y)$. Note that this naive alternating minimization approach with optimal accelerated rates only gives a iteration complexity bound of $O(\sqrt{\kappa}\log(n_x/\epsilon))$ gradient steps per hyperparameter evaluation step, and would require $O(n_y\sqrt{\kappa}\log(n_y/\epsilon))$ hyperparameter evaluations (Nesterov, 2013). This gives a total iteration complexity of $O(n_y\kappa\log(n/\epsilon)^2)$. Note that we may also solve 1 by simply running zeroth order optimization only on (x, y) jointly, without any gradient access. However, this gives a runtime of $O(n\sqrt{\kappa}\log(1/\epsilon))$, which is worse when κ is small.

1.2 Our contributions

In this paper, we develop the theoretical foundations for explaining the empirical success of jointly optimizing a bivariate function of the form $f(x, y)$ and show theoretically that joint optimization should be done with an unbiased gradient estimate, implying runtime savings when equal emphasis on training and tuning is applied. To our knowledge, these are the first theoretical results for general joint optimization. For theory, we assume that f is convex and let κ be its condition number; however, from experimental evidence, we note this assumption seems unnecessary for fast convergence in practice.

Improving on the classical bound, we present a novel analysis of our Joint Descent (JD) algorithms that uses a careful balance of first order and zeroth order steps to show that we may achieve an accelerated iteration complexity of $O(n_y\sqrt{\kappa}\log(n_y/\epsilon))$, which is a factor of $\sqrt{\kappa}\log(n/\epsilon)$ better than our classical bound. Note that it was apriori unclear if acceleration was even possible due to lack of symmetry in x, y. Furthermore, our bound is essentially the same iteration complexity of optimization $f(x, y)$ over y only, meaning that our bound is optimal and we achieve training for free. We present an informal version of our theorem below.

Theorem 1 (Informal Restatement of Theorem 4)**.** *Let z_k be the iterates of running Accelerated Joint Descent (Algorithm 2) with stochastic gradient g_η. Then,*

$$\mathbb{E}\left[f(z_k)\right] - f(z^*) \leq (1-\gamma)^k\xi_0 + O(\eta^2\beta^2\sqrt{\kappa}(n_y)^2)$$

with $\gamma = \Omega((\sqrt{\kappa}n_y)^{-1})$ and $\xi_0 = O(poly(n))$. Therefore, running $k = O(n_y\sqrt{\kappa} poly\log(n/\epsilon))$ iterations of AJD guarantees $\mathbb{E}\left[f(z_k)\right] - f(x^*) \leq \epsilon$ with η sufficiently small.

Intuitively, our proposed JD algorithms will take a gradient step with probability p (training) and a zeroth-order step with probability $1 - p$ (tuning). In the case of hyperparameter tuning of models in practice, note that if the training and tuning datasets are the same, we recover the standard hyperparameter tuning approaches by setting $p \approx 1$ and taking a zeroth-order step sparingly. However, our theory seems to imply that $p = 1/2$ might lead to faster convergence, as our analysis of acceleration requires an unbiased gradient.

Although this may seem like an artifact of our analysis, we empirically observe that setting $p = 1/2$ is the most effective for function optimization, even when the number of hyperparameters is large and equal to the number of training variables. This is especially surprising because this goes against the intuition that the gradient step probability p should be decreasing when the number of hyperparameters increase, especially since the convergence rate for tuning hyperparameters is slower than that of training. This perhaps demonstrates the importance of an unbiased gradient estimates when applying acceleration methods and underscores the utility of veering away from traditional hyperparameter tuning strategies, which has p very close to 1.

2 PRELIMINARIES

We first define a few notations for the rest of the paper. The full gradient is denoted as $\nabla f(x,y) \in \mathbb{R}^{n_x+n_y}$ and we let the partial gradients $\nabla_x f(x,y) \in \mathbb{R}^{n_x}, \nabla_y f(x,y) \in \mathbb{R}^{n_y}$ be the vectors such that $[\nabla_x f(x,y), \nabla_y f(x,y)] = \nabla f(x,y)$ where $[v,w]$ denotes the concatenation of the two vectors v,w. We often let $z = [x,y]$. Let $\|\cdot\|$ denote the Euclidean norm, unless otherwise specified, and $\langle x,y\rangle$ be the Euclidean inner product. Note that we always assume that x,y are within the domain of f and that the optimization is done over a compact region with diameter $O(\text{poly}(n))$.

Definition 2. *We say that f is α-strongly convex for $\alpha > 0$ if $f(y) \geq f(x) + \langle \nabla f(x), y-x\rangle + \frac{\alpha}{2}\|y-x\|^2$ for all x,y. We say that f is β-smooth for $\beta > 0$ if $f(y) \leq f(x) + \langle \nabla f(x), y-x\rangle + \frac{\beta}{2}\|y-x\|^2$ for all x,y.*

In this paper, we focus on the case where f is β-smooth and α-strongly convex for the theoretical guarantees. If $f(x,y)$ be β-smooth and α-strong convex, then we denote $\kappa = \beta/\alpha$ as its condition number.

2.1 ZEROTH VS FIRST ORDER GUARANTEES

Intuitively, the zeroth and first order rates are derived from the guaranteed progress they make at each step. With vanilla gradient descent, we have the following descent guarantee for minimizing a univariate smooth, convex function f. Let $x^+ = x - (1/\beta)\nabla f(x)$ be our next iterate after a gradient step. By simply combining the β-smoothness with the definition of x^+, we have

$$f(x^+) \leq f(x) - \frac{1}{\beta}\langle \nabla f(x), \nabla f(x)\rangle + \frac{\beta}{2}\|\nabla f(x)\|^2 \leq f(x) - \frac{1}{\beta}\|\nabla f(x)\|^2$$

From a classical analysis, this, combined by properties of α-strong convexity, leads to a convergence bound of using $O(\kappa \log(1/\epsilon))$ iterations to reach ϵ of the optimum (Nesterov, 2013). By applying momentum, we may accelerate our convergence to an iteration complexity of $O(\sqrt{\kappa}\log(1/\epsilon))$ which has an optimal square root dependence on the condition number. In general, applying acceleration requires a delicate linear combination between current and previous iterates and can be seen as a delicate coupling of primal and dual perspectives given by the strong convexity and smoothness properties (Allen-Zhu & Orecchia, 2014; Bubeck et al., 2015).

For zeroth order optimization, due to the lack of access to a gradient direction, many algorithms instead generates some random movement direction and decides the magnitude of movement based on some function valuation. Specifically, our next iterate becomes $x^+ = x - hu$, where u is a random unit direction and h is some stepsize parameter. Note that if $f(x^+) > f(x)$, most zeroth order methods will flip the movement direction and simply go to $x^+ = x + hu$. By repeating a similar analysis, we derive the following descent guarantee:

$$\mathbb{E}[f(x^+)] \leq f(x) - h\mathbb{E}\left[|\langle \nabla f(x), u\rangle|\right] + \frac{\beta}{2}h^2$$

If we let n be the dimension of x, note that $\mathbb{E}\left[|\langle \nabla f(x), u\rangle|\right] \approx \frac{1}{\sqrt{n}}\|\nabla f(x)\|$ and so our methods are about less efficient by an order of n because now our stepsize must be chosen to be $h = O(\|\nabla f(x)\|/\sqrt{n})$:

$$\mathbb{E}[f(x^+)] \leq f(x) - \frac{1}{\beta n}\|\nabla f(x)\|^2$$

Inevitably, this factor of n slowdown shows up the in the convergence rate of zeroth-order methods since from an information-theoretic perspective, a gradient evaluation is $O(n)$ more information than a function evaluation. However, function evaluations tend to be cheaper and we can still accelerate to achieve an iteration complexity of $O(n\sqrt{\kappa}\log(1/\epsilon))$ iterations.

Algorithm 1: Joint Descent (JD)

Input : function: $f(x, y)$, initial points: x_0, y_0, probability of gradient step: $0 < p < 1$, min/max search radius r, R

$z_0 = (x_0, y_0)$
for $k = 0, \ldots, T$ **do**
  With probability p, apply gradient step; otherwise, apply gradientless step:
  Gradient Step: $x_{k+1} = x_k - \beta^{-1}\nabla_x f(x, y)$
  Gradientless Step: For $i \in [1, 2, ..., \log_2(R/r)]$, let $r_i = 2^i r$ and sample $g_i \sim N(0, r_i\mathbf{I})$. Let $Y = \{y\} \cup \bigcup_i \{y + g_i\}$. Set $y_{k+1} = \arg\min_{y \in Y} f(x_k, y)$
end
return $z_T = (x_T, y_T)$

3 Joint optimization convergence rates

3.1 Vanilla rates

We first introduce convergence rates for vanilla joint optimization and proceed with an informal calculation. The main idea behind joint optimization is to use a probabilistic combination of zeroth and first order optimization, instead of some deterministic tradeoff or criteria. Let us take a gradient step with respect to x probability p and take a zeroth step with respect to y otherwise. Alternatively, we can view each iteration as a stochastic gradient step given by:

$$g_\eta^p(z) = \begin{cases} \nabla_x f(x, y) & w.p.\ p \\ \frac{f(x, y+\eta u) - f(x, y)}{\eta} u & w.p.\ 1 - p \end{cases}$$

Note that by linearity of expectation and our preliminary descent guarantees, our new descent guarantee becomes:

$$\mathbb{E}[f(z^+)] \leq f(z) - (1-p)\frac{1}{n_y \beta}\|\nabla_y f(x, y)\|^2 - (p)\frac{1}{\beta}\|\nabla_x f(x, y)\|^2$$

How should p be chosen? It seems intuitive that because the zeroth order step tends to be making less progress in general, the algorithm should lean towards performing more zeroth order steps. Indeed, by setting $p = \frac{1}{n_y+1}$, we balance the two descent guarantees to get:

$$\mathbb{E}[f(z^+)] \leq f(z) - \frac{1}{(n_y+1)\beta}\left(\|\nabla_y f(x, y)\|^2 + \|\nabla_x f(x, y)\|^2\right) \leq f(z) - \frac{1}{\beta(n_y+1)}\|\nabla f(z)\|^2$$

Therefore, we expect to be able to achieve an optimal runtime of $O(n_y\sqrt{\kappa} * \text{poly}\log(1/\epsilon))$, which is surprisingly the same order as the runtime of optimizing over y only.

We introduce our vanilla joint optimization algorithm Joint Descent (Algorithm 1), which combines standard gradient descent with the Gradientless Descent zeroth order optimization of Golovin et al. (2019). For now, we avoid gradient-approximating zeroth order techniques because 1) they tend to be less robust and 2) their analysis is complicated by the induced numerical approximation errors. Instead, Gradientless Descent uses a binary search over radii to find the optimal stepsize.

Theorem 3. *Let $\frac{1}{n_y+1} \leq p \leq 0.5$ and $R =$ poly(n) is the diameter of the search space and $r = \frac{\epsilon}{\beta\sqrt{n_y}}$. Then, running $k = O(n_y \kappa \text{poly} \log(n/\epsilon))$ iterations of Joint Descent (Algorithm 1) guarantees $\mathbb{E}[f(z_k)] - f(z^*) \leq O(\epsilon)$, where z_k is the k-th iterate.*

3.2 Acceleration with unbiased gradients

To achieve provably accelerated rates, we often need a careful balancing of optimization steps and this balancing is complicated further in joint optimization. Specifically, for the analysis to hold, we must first ensure that we have a unbiased estimate of the gradient of the function, as the gradient

Algorithm 2: Accelerated Joint Descent (AJD)

Input : function: $f(x, y)$, initial points: x_0, y_0
$h = (8\beta(n_y + 4))^{-1}$, $\theta = h^2\beta$, $a = \sqrt{\alpha\theta}$,
$z_0 = (x_0, y_0), v_0 = z_0$
for $k = 0, \ldots, T$ **do**
  Linear Combination: $w_k = \frac{1}{1+a} z_k + \frac{a}{1+a} v_k$
  Stochastic Step: Calculate $g_\eta(w_k)$ for random $u \sim N(0, \mathbf{I})$
  Update: $z_{k+1} = w_k - h g_\eta(w_k), v_{k+1} = (1-a)v_k + a w_k - \frac{\theta}{a} g_\eta(w_k)$
end
return $z_T = (x_T, y_T)$

plays an important role in getting the right stepsize for acceleration. However, there are two main issues: 1) the joint optimization framework often leads to a biased gradient estimate and 2) we can only approximate $\nabla_y f(x, y)$.

The first issue can be solved by simply choosing the probability of gradient step $p = 0.5$ and applying the unbiased stochastic gradient $g_\eta^{0.5}(z)$ but this seems to be a suboptimal choice of p via the unaccelerated analysis of Theorem 3. Surprisingly however, our experiments seems to support the hypothesis that unbiased gradient estimates play a greater importance than choosing the optimal tradeoff for a better descent guarantee. As our experiments show, the optimal choice of p even in the unaccelerated regime tends to be around $p = 1/2$.

For the second issue, we use the notion of Gaussian smoothing introduced in Nesterov & Spokoiny (2011) and relate the gradient of the Gaussian smoothed function to the approximate gradient estimate. Specifically, for a bi-variate function $f(x, y)$, we define the *Gaussian smoothing* of f with respect to y and stepsize η as:

$$f_\eta(x, y) = \int f(x, y + \eta u) \frac{1}{P} e^{-\|u\|^2/2}\, du$$

where $P = \int e^{-\|u\|^2/2}\, du$ is the appropriate normalizing constant for the Gaussian density.

The main motivation for introducing Gaussian smoothing is because it relates the expected approximate gradient of $f(x, y)$ with respect to y to the exact gradient of $f_\eta(x, y)$. Note that we may rewrite our smoothed function with a new integration variable $z = y + \eta u$ and since our functions are continuous in y, z, we may differentiate to conclude that

$$\begin{aligned} \nabla_y f_\eta(x, y) &= \int f(x, z) \frac{1}{\eta^{n_y+2} P} e^{-\|z-y\|^2/2\eta^2} (z - y)\, dz \\ &= \int \left[\frac{f(x, y + \eta u) - f(x, y)}{\eta} u \right] \frac{1}{P} e^{-\|u\|^2/2}\, du \end{aligned}$$

where the first line follows from differentiation with respect to y and the second line from definition of z and $\int u e^{-\|u\|^2/2}\, du = 0$. Therefore, if we let our stochastic gradient be $g_\eta = 2 g_\eta^{0.5}(z)$, then it is unbiased for the smoothed function: $\mathbb{E}\left[g_\eta(z)\right] = \nabla f_\eta(z)$.

By equally balancing zeroth-order and first-order steps and analyzing the descent guarantees on f_η, we are able to effectively accelerate the optimization and derive strong theoretical guarantees. Note that since f_η is a convex combination of f, it follows that f_η is β-smooth and α-strongly convex. Furthermore, we can use the fact that $f_\eta \approx f$ to translate guarantees back to the original function.

Theorem 4. *Let z_k be the iterates of running Accelerated Joint Descent (Algorithm 2) with stochastic gradient $g_\eta = 2g_\eta^{0.5}$. Then,*

$$\mathbb{E}\left[f(z_k)\right] - f(z^*) \le (1-\gamma)^k \xi_0 + 9\eta^2\beta^2\sqrt{\kappa}(n_y + 4)^2$$

with $\gamma = \Omega((\sqrt{\kappa} n_y)^{-1})$ and $\xi_0 = \frac{\alpha}{2}\|z_0 - z^*\|^2 + f(z_0) - f(z^*) + \eta^2 \beta n_y$. Therefore, running $k = O(n_y \sqrt{\kappa} poly \log(n/\epsilon))$ iterations of AJD guarantees $\mathbb{E}\left[f(z_k)\right] - f(x^*) \le \epsilon$ when $\eta^2 = O(\epsilon(\beta^2 n_y^2 \sqrt{\kappa})^{-1})$.

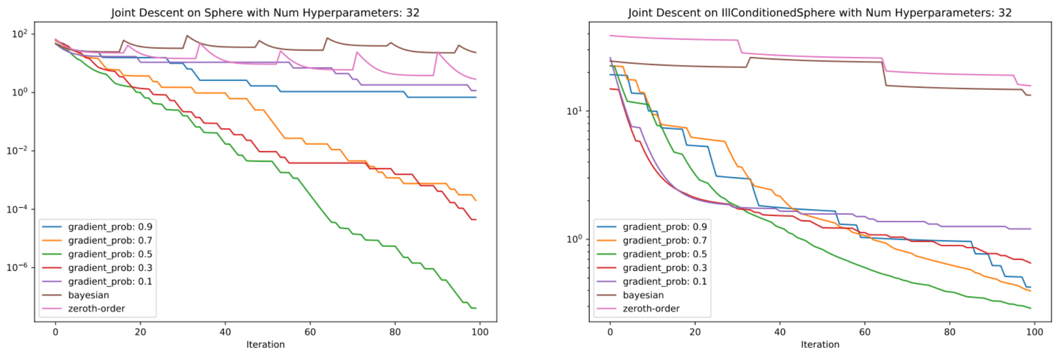

Figure 1: Convergence plot for the Sphere and Ill-Conditioned Sphere with 30 training variables and 32 hyperparameters, with varying algorithms and JD with varying gradient probabilities. Note that Joint Descent significantly outperforms the traditional algorithms that alternate between training and tuning and thereby produce a sawtooth-like training curve. Furthermore, the optimal gradient probability is around 0.5, even when the number of hyperparameters is 32.

4 Experiments

The goal of our experiments is to empirically support our theoretical contributions and demonstrate the utility of joint descent in machine learning. In this paper, we focus on the former and demonstrate the utility of an unbiased gradient estimate since the latter has been established in past papers. Note that hese empirical results together show that although convexity is not realistic in practice, the theoretical computational gains translate over to practical nonconvex settings.

4.1 Sphere function

We empirically study the effect of gradient probability, ill-conditioning and acceleration on our performance with simple convex functions. According to the theory, it seems unclear how to choose the probability p of choosing a gradient step in each iteration. From the descent guarantee analysis of JD (Theorem 3), the gradient probability should be inversely proportional to the number of hyperparameters n_y. However, if we view choosing our descent step in x or y as a stochastic gradient, this would induce a biased gradient estimate.

We run experiments on a quadratic Sphere function, specifically $f(x, y) = \|x\|^2 + \|y\|^2$, and its ill-conditioned version by simply scaling the quadratic function by in each coordinate by s_i, where s_i ranges linearly from 1 to n. Furthermore, we randomly rotate and shift the function. We run JD with gradient probability $p \in [0.1, 0.3, 0.5, 0.7, 0.9]$ and its accelerated version AJD. We also include standard alternating algorithms such as Bayesian optimization (Frazier, 2018) and zeroth-order optimization (Nesterov & Spokoiny, 2011), where in each round of hyperparameter selection, there is a full optimization cycle on the training variables x, which is randomly initialized between two hyperparameter settings. In the alternating case, we end an optimization cycle on x by detecting relative convergence of 0.01 and we select the next hyperparameters using either Bayesian methods or zeroth-order optimization.

From our plots of the objective evaluation $f(x, y)$ at each iteration, we notice surprisingly that keeping gradient probability at around $p = 0.5$ seems to perform the best, implying that an unbiased gradient estimator helps to accelerate optimization more than balancing first-order vs zeroth-order convergence rates (Figure 1). This also perhaps implies that zeroth-order optimization iteration bounds are overly pessimistic. As predicted from our theory, when acceleration is applied to JD, we see that setting gradient probability $p = 0.5$ is crucial for optimal performance (Figure 2).

Lastly, our joint optimization significantly outperforms traditional tuning algorithms, such as Bayesian optimization and zeroth-order optimization, reaching a better objective even before second selection of hyperparameters occurs. Our plots also confirm that optimization algorithms on ill-conditioned functions tend to do significantly worse, although acceleration does aid performance. Full convergence plots of JD and AJD on various settings are given in the supplementary material.

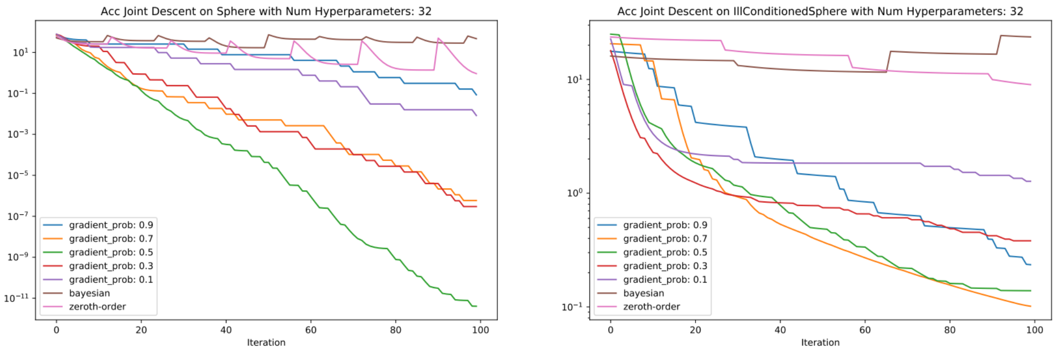

Figure 2: Objective convergence plot for Accelerated Joint Descent with the same setting as Fig 1. Note that acceleration outperforms the non-accelerated version and the optimal gradient probability is still around 0.5.

4.2 MNIST DATASET

We demonstrate the utility of joint descent on the full MNIST dataset for optimizing training accuracy (LeCun et al., 1998). Our model is a simple two-layer feedforward neural network that takes the flattened image input of 784 dimensions with 30 hidden nodes and 10 linear output nodes. The prediction is simply the argmax over all 10 nodes and we train the network with sparse cross entropy loss. The hyperparameters are the learning rate and the activation function, which is set to be $\sigma(x) = ReLU(x, \alpha) + \beta * eLU(x)$, where $ReLU(x, \alpha)$ is the ReLU unit with slope α for $x < 0$ and $eLU(x)$ denotes the eLU unit. Note that to incorporate learning rate into the loss, our objective is of the form $f(x, (\sigma, \eta)) = f_\sigma(x - \eta \nabla f_\sigma(x))$, where η is the learning rate hyperparameter. For the initialization, we set $\alpha = 1$, which implies a linear activation, and we wish to see if joint descent can learn to set α close to zero to emulate the performance of the ReLU network. Note that some common hyperparameters, such as batch size and optimizer choice, are discrete and not tuned.

From figure 3, we see that JD ends up with a competitive accuracy of 96%, matching that of a ReLU network after only one training run, implying significant computational savings since a full alternating strategy, such as Bayesian optimization, will cycle many training runs for different settings of hyperparameters. Note that after simply 500 iterations of combined training and tuning, the network learns to set the ReLU slope to be close to 0 and outperform the linear model. Also, we note that JD learns to reduce the learning rate upon convergence. The full plots for all hyperparameters are given in the supplementary material.

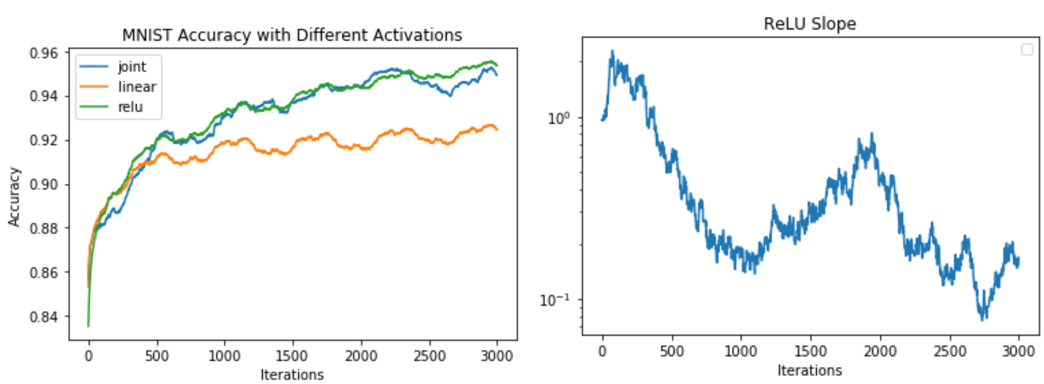

Figure 3: Training accuracy for Joint Descent for the MNIST dataset with hyperparameters: learning rate, ReLU slope, eLU cofficient. The ReLU slope is set to 1, implying a linear activation, but the model learns to set the ReLU slope to be close to 0 to achieve competitive performance.

REFERENCES

Zeyuan Allen-Zhu and Lorenzo Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent. *arXiv preprint arXiv:1407.1537*, 2014.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pp. 3981–3989, 2016.

Anne Auger and Nikolaus Hansen. A restart cma evolution strategy with increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pp. 1769–1776. IEEE, 2005.

Krishnakumar Balasubramanian and Saeed Ghadimi. Zeroth-order (non)-convex stochastic optimization via conditional gradient and gradient updates. In *Advances in Neural Information Processing Systems*, pp. 3455–3464, 2018.

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Samuel H Brooks. A discussion of random methods for seeking maxima. *Operations research*, 6(2): 244–251, 1958.

Sébastien Bubeck, Yin Tat Lee, and Mohit Singh. A geometric alternative to nesterov's accelerated gradient descent. *arXiv preprint arXiv:1506.08187*, 2015.

Krzysztof Choromanski, Mark Rowland, Vikas Sindhwani, Richard E Turner, and Adrian Weller. Structured evolution with compact architectures for scalable policy optimization. *arXiv preprint arXiv:1804.02395*, 2018.

Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 874–883. JMLR. org, 2017.

John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, 2015.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 385–394. Society for Industrial and Applied Mathematics, 2005.

Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

Daniel Golovin, John Karro, Greg Kochanski, Chansoo Lee, Xingyou Song, et al. Gradientless descent: High-dimensional zeroth-order optimization. *arXiv preprint arXiv:1911.06317*, 2019.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Advances in neural information processing systems*, 3(06), 2014.

Eduard Gorbunov, Adel Bibi, Ozan Sener, El Houcine Bergou, and Peter Richtárik. A stochastic derivative free optimization method with momentum. *arXiv preprint arXiv:1905.13278*, 2019.

Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA, 2009.

Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pp. 65–93. Elsevier, 1992.

Le Hou, Dimitris Samaras, Tahsin M Kurc, Yi Gao, and Joel H Saltz. Convnets with smooth adaptive activation functions for regression. *Proceedings of machine learning research*, 54:430, 2017.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Tomer Lancewicki and Selcuk Kopru. Automatic and simultaneous adjustment of learning rate and momentum for stochastic gradient descent. *arXiv preprint arXiv:1908.07607*, 2019.

Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998. *URL http://yann. lecun. com/exdb/mnist*, 10:34, 1998.

Shizhong Liao and Lei Jia. Simultaneous tuning of hyperparameter and parameter for support vector machines. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 162–172. Springer, 2007.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018a.

Sijia Liu, Bhavya Kailkhura, Pin-Yu Chen, Paishun Ting, Shiyu Chang, and Lisa Amini. Zeroth-order stochastic variance reduction for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pp. 3727–3737, 2018b.

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International conference on machine learning*, pp. 2952–2960, 2016.

Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *arXiv preprint arXiv:1903.03088*, 2019.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pp. 2113–2122, 2015.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. Technical report, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2011.

Atsushi Nitanda. Stochastic proximal gradient descent with acceleration techniques. In *Advances in Neural Information Processing Systems*, pp. 1574–1582, 2014.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 506–519. ACM, 2017.

Ohad Shamir. An optimal algorithm for bandit and zero-order convex optimization with two-point feedback. *Journal of Machine Learning Research*, 18(52):1–11, 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423, 2009.