Probabilistic Line Searches
for Stochastic Optimization

Maren Mahsereci and Philipp Hennig
Max Planck Institute for Intelligent Systems
Spemannstraße 38, 72076 Tübingen, Germany
[mmahsereci|phennig]@tue.mpg.de

Abstract

In deterministic optimization, line searches are a standard tool ensuring stability and efficiency. Where only stochastic gradients are available, no direct equivalent has so far been formulated, because uncertain gradients do not allow for a strict sequence of decisions collapsing the search space. We construct a probabilistic line search by combining the structure of existing deterministic methods with notions from Bayesian optimization. Our method retains a Gaussian process surrogate of the univariate optimization objective, and uses a probabilistic belief over the Wolfe conditions to monitor the descent. The algorithm has very low computational cost, and no user-controlled parameters. Experiments show that it effectively removes the need to define a learning rate for stochastic gradient descent.

1   Introduction

Stochastic gradient descent (SGD) [1] is currently the standard in machine learning for the optimization of highly multivariate functions if their gradient is corrupted by noise. This includes the online or batch training of neural networks, logistic regression [2, 3] and variational models [e.g. 4, 5, 6]. In all these cases, noisy gradients arise because an exchangeable loss-function $\mathcal{L}(x)$ of the optimization parameters $x \in \mathbb{R}^D$, across a large dataset $\{d_i\}_{i=1\,...,M}$, is evaluated only on a subset $\{d_j\}_{j=1,...,m}$:

$$\mathcal{L}(x) := \frac{1}{M} \sum_{i=1}^{M} \ell(x, d_i) \approx \frac{1}{m} \sum_{j=1}^{m} \ell(x, d_j) =: \hat{\mathcal{L}}(x) \qquad m \ll M. \tag{1}$$

If the indices j are i.i.d. draws from $[1, M]$, by the Central Limit Theorem, the error $\hat{\mathcal{L}}(x) - \mathcal{L}(x)$ is unbiased and approximately normal distributed. Despite its popularity and its low cost per step, SGD has well-known deficiencies that can make it inefficient, or at least tedious to use in practice. Two main issues are that, first, the gradient itself, even without noise, is not the optimal search *direction*; and second, SGD requires a *step size* (learning rate) that has drastic effect on the algorithm's efficiency, is often difficult to choose well, and virtually never optimal for each individual descent step. The former issue, adapting the search direction, has been addressed by many authors [see 7, for an overview]. Existing approaches range from lightweight 'diagonal preconditioning' approaches like ADAGRAD [8] and 'stochastic meta-descent'[9], to empirical estimates for the natural gradient [10] or the Newton direction [11], to problem-specific algorithms [12], and more elaborate estimates of the Newton direction [13]. Most of these algorithms also include an auxiliary adaptive effect on the learning rate. And Schaul et al. [14] recently provided an estimation method to explicitly adapt the learning rate from one gradient descent step to another. None of these algorithms change the size of the *current* descent step. Accumulating statistics across steps in this fashion requires some conservatism: If the step size is initially too large, or grows too fast, SGD can become unstable and 'explode', because individual steps are not checked for robustness at the time they are taken.
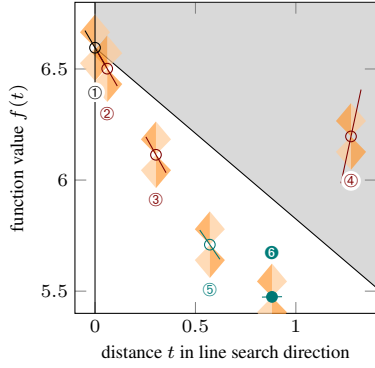
Figure 1: Sketch: The task of a **classic line search** is to tune the step taken by a optimization algorithm along a univariate search direction. The search starts at the endpoint ① of the previous line search, at $t = 0$. A sequence of exponentially growing extrapolation steps ②,③,④ finds a point of positive gradient at ④. It is followed by interpolation steps ⑤,❻ until an acceptable point ❻ is found. Points of insufficient decrease, above the line $f(0) + c_1 t f'(0)$ (gray area) are excluded by the Armijo condition W-I, while points of steep gradient (orange areas) are excluded by the curvature condition W-II (weak Wolfe conditions in solid orange, strong extension in lighter tone). Point ❻ is the first to fulfil both conditions, and is thus accepted.

The principally same problem exists in deterministic (noise-free) optimization problems. There, providing stability is one of several tasks of the *line search* subroutine. It is a standard constituent of algorithms like the classic nonlinear conjugate gradient [15] and BFGS [16, 17, 18, 19] methods [20, §3].[1] In the noise-free case, line searches are considered a solved problem [20, §3]. But the methods used in deterministic optimization are not stable to noise. They are easily fooled by even small disturbances, either becoming overly conservative or failing altogether. The reason for this brittleness is that existing line searches take a sequence of hard decisions to shrink or shift the search space. This yields efficiency, but breaks hard in the presence of noise. Section 3 constructs a probabilistic line search for noisy objectives, stabilizing optimization methods like the works cited above. As line searches only change the length, not the direction of a step, they could be used in combination with the algorithms adapting SGD's direction, cited above. The algorithm presented below is thus a complement, not a competitor, to these methods.

2 Connections

2.1 Deterministic Line Searches

There is a host of existing line search variants [20, §3]. In essence, though, these methods explore a univariate domain 'to the right' of a starting point, until an 'acceptable' point is reached (Figure 1). More precisely, consider the problem of minimizing $\mathcal{L}(x) : \mathbb{R}^D \rightarrow \mathbb{R}$, with access to $\nabla\mathcal{L}(x) : \mathbb{R}^D \rightarrow \mathbb{R}^D$. At iteration i, some 'outer loop' chooses, at location x_i, a search direction $s_i \in \mathbb{R}^D$ (e.g. by the BFGS rule, or simply $s_i = -\nabla\mathcal{L}(x_i)$ for gradient descent). It will *not* be assumed that s_i has unit norm. The line search operates along the univariate domain $x(t) = x_i + t s_i$ for $t \in \mathbb{R}_+$. Along this direction it collects scalar function values and projected gradients that will be denoted $f(t) = \mathcal{L}(x(t))$ and $f'(t) = s_i^\mathsf{T}\nabla\mathcal{L}(x(t)) \in \mathbb{R}$. Most line searches involve an initial extrapolation phase to find a point t_r with $f'(t_r) > 0$. This is followed by a search in $[0, t_r]$, by interval nesting or by interpolation of the collected function and gradient values, e.g. with cubic splines.[2]

2.1.1 The Wolfe Conditions for Termination

As the line search is only an auxiliary step within a larger iteration, it need not find an exact root of f'; it suffices to find a point 'sufficiently' close to a minimum. The *Wolfe [21] conditions* are a widely accepted formalization of this notion; they consider t acceptable if it fulfills

$$f(t) \leq f(0) + c_1 t f'(0) \quad \text{(W-I)} \qquad \text{and} \qquad f'(t) \geq c_2 f'(0) \quad \text{(W-II)}, \tag{2}$$

using two constants $0 \leq c_1 < c_2 \leq 1$ chosen by the designer of the line search, not the user. W-I is the *Armijo [22]*, or *sufficient decrease* condition. It encodes that acceptable functions values should lie below a linear extrapolation line of slope $c_1 f'(0)$. W-II is the *curvature condition*, demanding

---

[1] In these algorithms, another task of the line search is to guarantee certain properties of surrounding estimation rule. In BFGS, e.g., it ensures positive definiteness of the estimate. This aspect will not feature here.

[2] This is the strategy in `minimize.m` by C. Rasmussen, which provided a model for our implementation. At the time of writing, it can be found at `http://learning.eng.cam.ac.uk/carl/code/minimize/minimize.m`
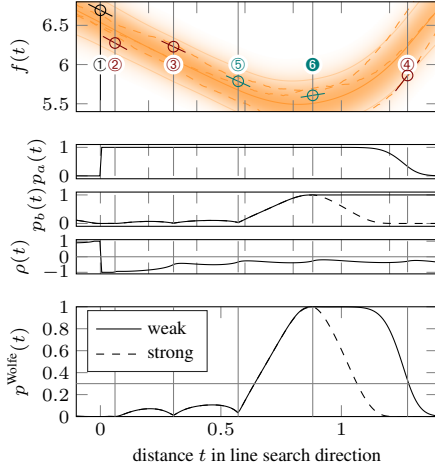
Figure 2: Sketch of a **probabilistic line search**. As in Fig. 1, the algorithm performs extrapolation (②,③,④) and interpolation (⑤,❻), but receives unreliable, noisy function and gradient values. These are used to construct a GP posterior (top. solid posterior mean, thin lines at 2 standard deviations, local pdf marginal as shading, three dashed sample paths). This implies a bivariate Gaussian belief (§3.3) over the validity of the weak Wolfe conditions (middle three plots. $p_a(t)$ is the marginal for W-I, $p_b(t)$ for W-II, $\rho(t)$ their correlation). Points are considered acceptable if their joint probability $p^{\text{Wolfe}}(t)$ (bottom) is above a threshold (gray). An approximation (§3.3.1) to the strong Wolfe conditions is shown dashed.

a decrease in slope. The choice $c_1 = 0$ accepts any value below $f(0)$, while $c_1 = 1$ rejects all points for convex functions. For the curvature condition, $c_2 = 0$ only accepts points with $f'(t) \geq 0$; while $c_2 = 1$ accepts any point of greater slope than $f'(0)$. W-I and W-II are known as the *weak* form of the Wolfe conditions. The *strong* form replaces W-II with $|f'(t)| \leq c_2|f'(0)|$ (W-IIa). This guards against accepting points of low function value but large positive gradient. Figure 1 shows a conceptual sketch illustrating the typical process of a line search, and the weak and strong Wolfe conditions. The exposition in §3.3 will initially focus on the weak conditions, which can be precisely modeled probabilistically. Section 3.3.1 then adds an approximate treatment of the strong form.

2.2   Bayesian Optimization

A recently blossoming sample-efficient approach to global optimization revolves around modeling the objective f with a probability measure $p(f)$; usually a Gaussian process (GP). Searching for extrema, evaluation points are then chosen by a utility functional $u[p(f)]$. Our line search borrows the idea of a Gaussian process surrogate, and a popular utility, *expected improvement* [23]. Bayesian optimization methods are often computationally expensive, thus ill-suited for a cost-sensitive task like a line search. But since line searches are governors more than information extractors, the kind of sample-efficiency expected of a Bayesian optimizer is not needed. The following sections develop a lightweight algorithm which adds only minor computational overhead to stochastic optimization.

3   A Probabilistic Line Search

We now consider minimizing $y(t) = \hat{\mathcal{L}}(x(t))$ from Eq. (1). That is, the algorithm can access only noisy function values and gradients y_t, y'_t at location t, with Gaussian likelihood

$$p(y_t, y'_t \mid f) = \mathcal{N}\left(\begin{bmatrix} y_t \\ y'_t \end{bmatrix}; \begin{bmatrix} f(t) \\ f'(t) \end{bmatrix}, \begin{bmatrix} \sigma_f^2 & 0 \\ 0 & \sigma_{f'}^2 \end{bmatrix}\right). \tag{3}$$

The Gaussian form is supported by the Central Limit argument at Eq. (1), see §3.4 regarding estimation of the variances $\sigma_f^2, \sigma_{f'}^2$. Our algorithm has three main ingredients: A robust yet lightweight Gaussian process surrogate on $f(t)$ facilitating analytic optimization; a simple Bayesian optimization objective for exploration; and a probabilistic formulation of the Wolfe conditions as a termination criterion.

3.1   Lightweight Gaussian Process Surrogate

We model information about the objective in a probability measure $p(f)$. There are two requirements on such a measure: First, it must be robust to irregularity of the objective. And second, it must allow analytic computation of discrete candidate points for evaluation, because a line search should not call yet another optimization subroutine itself. Both requirements are fulfilled by a once-integrated Wiener process, i.e. a zero-mean Gaussian process prior $p(f) = \mathcal{GP}(f; 0, k)$ with covariance function

$$k(t, t') = \theta^2 \left[\nicefrac{1}{3}\min^3(\tilde{t}, \tilde{t}') + \nicefrac{1}{2}|t - t'|\min^2(\tilde{t}, \tilde{t}') \right]. \tag{4}$$

3

Here $\tilde{t} := t + \tau$ and $\tilde{t}' := t' + \tau$ denote a shift by a constant $\tau > 0$. This ensures this kernel is positive semi-definite, the precise value τ is irrelevant as the algorithm only considers positive values of t (our implementation uses $\tau = 10$). See §3.4 regarding the scale θ^2. With the likelihood of Eq. (3), this prior gives rise to a GP posterior whose mean function is a cubic spline[3] [25]. We note in passing that regression on f and f' from N observations of pairs (y_t, y'_t) can be formulated as a filter [26] and thus performed in $\mathcal{O}(N)$ time. However, since a line search typically collects < 10 data points, generic GP inference, using a Gram matrix, has virtually the same, low cost.

Because Gaussian measures are closed under linear maps [27, §10], Eq. (4) implies a Wiener process (linear spline) model on f':

$$p(f; f') = \mathcal{GP}\left(\begin{bmatrix} f \\ f' \end{bmatrix}; \mathbf{0}, \begin{bmatrix} k & k^\partial \\ \partial k & \partial k^\partial \end{bmatrix}\right), \tag{5}$$

with (using the indicator function $\mathbb{I}(x) = 1$ if x, else 0)

$$\partial^i k \partial^j = \frac{\partial^{i+j} k(t, t')}{\partial t^i \partial t'^j}, \quad \text{thus} \quad \begin{aligned} k^\partial(t, t') &= \theta^2 \left[\mathbb{I}(t < t') t^2/2 + \mathbb{I}(t \geq t')(tt' - t'^2/2)\right] \\ \partial k(t, t') &= \theta^2 \left[\mathbb{I}(t' < t) t'^2/2 + \mathbb{I}(t' \geq t)(tt' - t^2/2)\right] \\ \partial k^\partial(t, t') &= \theta^2 \min(t, t') \end{aligned} . \tag{6}$$

Given a set of evaluations $(\boldsymbol{t}, \boldsymbol{y}, \boldsymbol{y}')$ (vectors, with elements t_i, y_{t_i}, y'_{t_i}) with independent likelihood (3), the posterior $p(f \mid \boldsymbol{y}, \boldsymbol{y}')$ is a GP with posterior mean μ and covariance and $\tilde{k}$ as follows:

$$\mu(t) = \underbrace{\begin{bmatrix} k_{\boldsymbol{t}t} \\ \partial k_{\boldsymbol{t}t} \end{bmatrix}^\mathsf{T} \left(\begin{bmatrix} k_{\boldsymbol{t}\boldsymbol{t}} + \sigma_f^2 \boldsymbol{I} & k^\partial_{\boldsymbol{t}\boldsymbol{t}} \\ \partial k_{\boldsymbol{t}\boldsymbol{t}} & \partial k^\partial_{\boldsymbol{t}\boldsymbol{t}} + \sigma_{f'}^2 \boldsymbol{I} \end{bmatrix}\right)^{-1}}_{=: \boldsymbol{g}^\mathsf{T}(t)} \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}' \end{bmatrix}, \quad \tilde{k}(t, t') = k_{tt'} - \boldsymbol{g}^\mathsf{T}(t) \begin{bmatrix} k_{\boldsymbol{t}t'} \\ \partial k_{\boldsymbol{t}t'} \end{bmatrix}. \tag{7}$$

The posterior marginal variance will be denoted by $\mathbb{V}(t) = \tilde{k}(t, t)$. To see that μ is indeed piecewise cubic (i.e. a cubic spline), we note that it has at most three non-vanishing derivatives[4], because

$$\partial^2 k(t, t') = \theta^2 \mathbb{I}(t \leq t')(t' - t) \qquad\qquad \partial^2 k^\partial(t, t') = \theta^2 \mathbb{I}(t \leq t')$$
$$\partial^3 k(t, t') = -\theta^2 \mathbb{I}(t \leq t') \qquad\qquad \partial^3 k^\partial(t, t') = 0. \tag{8}$$

This piecewise cubic form of μ is crucial for our purposes: having collected N values of f and f', respectively, all local minima of μ can be found analytically in $\mathcal{O}(N)$ time in a single sweep through the 'cells' $t_{i-1} < t < t_i$, $i = 1, \ldots, N$ (here $t_0 = 0$ denotes the start location, where (y_0, y'_0) are 'inherited' from the preceding line search. For typical line searches $N < 10$, c.f. §4). In each cell, $\mu(t)$ is a cubic polynomial with at most one minimum in the cell, found by a trivial quadratic computation from the three scalars $\mu'(t_i), \mu''(t_i), \mu'''(t_i)$. This is in contrast to other GP regression models—for example the one arising from a Gaussian kernel—which give more involved posterior means whose local minima can be found only approximately. Another advantage of the cubic spline interpolant is that it does not assume the existence of higher derivatives (in contrast to the Gaussian kernel, for example), and thus reacts robustly to irregularities in the objective.

In our algorithm, after each evaluation of (y_N, y'_N), we use this property to compute a short list of *candidates* for the next evaluation, consisting of the $\leq N$ local minimizers of $\mu(t)$ and one additional *extrapolation* node at $t_{\max} + \alpha$, where $t_{\max}$ is the currently largest evaluated t, and α is an extrapolation step size starting at $\alpha = 1$ and doubled after each extrapolation step.

3.2 Choosing Among Candidates

The previous section described the construction of $< N + 1$ discrete candidate points for the next evaluation. To decide at which of the candidate points to actually call f and f', we make use of a popular utility from Bayesian optimization. *Expected improvement* [23] is the expected amount,

---

[3]Eq. (4) can be generalized to the 'natural spline', removing the need for the constant τ [24, §6.3.1]. However, this notion is ill-defined in the case of a single observation, which is crucial for the line search.

[4]There is no well-defined probabilistic belief over f'' and higher derivatives—sample paths of the Wiener process are almost surely non-differentiable almost everywhere [28, §2.2]. But $\mu(t)$ is always a member of the reproducing kernel Hilbert space induced by k, thus piecewise cubic [24, §6.1].
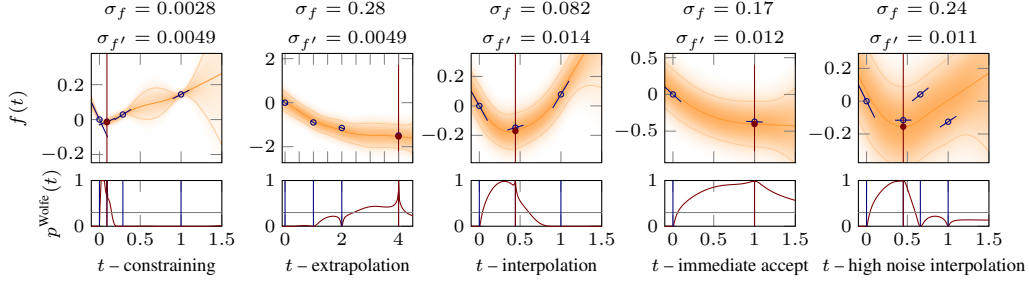
Figure 3: Curated snapshots of line searches (from MNIST experiment, §4), showing variability of the objective's shape and the decision process. Top row: GP posterior and evaluations, bottom row: approximate p^{Wolfe} over strong Wolfe conditions. Accepted point marked red.

under the GP surrogate, by which the function $f(t)$ might be smaller than a 'current best' value η (we set $\eta = \min_{i=0,\ldots,N}\{\mu(t_i)\}$, where t_i are observed locations),

$$
\begin{aligned}
u_{\text{EI}}(t) &= \mathbb{E}_{p(f_t \mid \boldsymbol{y}, \boldsymbol{y}')}[\min\{0, \eta - f(t)\}] \\
&= \frac{\eta - \mu(t)}{2}\left(1 + \text{erf}\,\frac{\eta - \mu(t)}{\sqrt{2\mathbb{V}(t)}}\right) + \sqrt{\frac{\mathbb{V}(t)}{2\pi}}\exp\left(-\frac{(\eta - \mu(t))^2}{2\mathbb{V}(t)}\right).
\end{aligned}
\tag{9}
$$

The next evaluation point is chosen as the candidate maximizing this utility, multiplied by the probability for the Wolfe conditions to be fulfilled, which is derived in the following section.

3.3 Probabilistic Wolfe Conditions for Termination

The key observation for a probabilistic extension of W-I and W-II is that they are positivity constraints on two variables a_t, b_t that are both linear projections of the (jointly Gaussian) variables f and f':

$$
\begin{bmatrix} a_t \\ b_t \end{bmatrix} = \begin{bmatrix} 1 & c_1 t & -1 & 0 \\ 0 & -c_2 & 0 & 1 \end{bmatrix}\begin{bmatrix} f(0) \\ f'(0) \\ f(t) \\ f'(t) \end{bmatrix} \geq 0.
\tag{10}
$$

The GP of Eq. (5) on f thus implies, at each value of t, a bivariate Gaussian distribution

$$
p(a_t, b_t) = \mathcal{N}\left(\begin{bmatrix} a_t \\ b_t \end{bmatrix}; \begin{bmatrix} m_t^a \\ m_t^b \end{bmatrix}, \begin{bmatrix} C_t^{aa} & C_t^{ab} \\ C_t^{ba} & C_t^{bb} \end{bmatrix}\right),
\tag{11}
$$

$$
\text{with} \qquad m_t^a = \mu(0) - \mu(t) + c_1 t \mu'(0) \qquad \text{and} \qquad m_t^b = \mu'(t) - c_2\mu'(0)
\tag{12}
$$

$$
\begin{aligned}
\text{and} \qquad C_t^{aa} &= \tilde{k}_{00} + (c_1 t)^2\, {}^\partial\tilde{k}_{00}^\partial + \tilde{k}_{tt} + 2[c_1 t(\tilde{k}_{00}^\partial - {}^\partial\tilde{k}_{0t}) - \tilde{k}_{0t}] \\
C_t^{bb} &= c_2^2\, {}^\partial\tilde{k}_{00}^\partial - 2c_2\, {}^\partial\tilde{k}_{0t}^\partial + {}^\partial\tilde{k}_{tt}^\partial \\
C_t^{ab} = C_t^{ba} &= -c_2(\tilde{k}_{00}^\partial + c_1 t\, {}^\partial\tilde{k}_{00}^\partial) + (1 + c_2)\,{}^\partial\tilde{k}_{0t} + c_1 t\, {}^\partial\tilde{k}_{0t}^\partial - \tilde{k}_{tt}^\partial.
\end{aligned}
\tag{13}
$$

The quadrant probability $p_t^{\text{Wolfe}} = p(a_t > 0 \wedge b_t > 0)$ for the Wolfe conditions to hold is an integral over a bivariate normal probability,

$$
p_t^{\text{Wolfe}} = \int_{-\frac{m_t^a}{\sqrt{C_t^{aa}}}}^{\infty}\int_{-\frac{m_t^b}{\sqrt{C_t^{bb}}}}^{\infty}\mathcal{N}\left(\begin{bmatrix} a \\ b \end{bmatrix}; \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho_t \\ \rho_t & 1 \end{bmatrix}\right)\,da\,db,
\tag{14}
$$

with correlation coefficient $\rho_t = C_t^{ab}/\sqrt{C_t^{aa}C_t^{bb}}$. It can be computed efficiently [29], using readily available code[5] (on a laptop, one evaluation of p_t^{Wolfe} cost about 100 microseconds, each line search requires < 50 such calls). The line search computes this probability for all evaluation nodes, after each evaluation. If any of the nodes fulfills the Wolfe conditions with $p_t^{\text{Wolfe}} > c_W$, greater than some threshold $0 < c_W \leq 1$, it is accepted and returned. If several nodes simultaneously fulfill this requirement, the t of the lowest $\mu(t)$ is returned. Section 3.4 below motivates fixing $c_W = 0.3$.

---

[5]e.g. http://www.math.wsu.edu/faculty/genz/software/matlab/bvn.m

3.3.1 Approximation for strong conditions:

As noted in Section 2.1.1, deterministic optimizers tend to use the strong Wolfe conditions, which use $|f'(0)|$ and $|f'(t)|$. A precise extension of these conditions to the probabilistic setting is numerically taxing, because the distribution over $|f'|$ is a non-central χ-distribution, requiring customized computations. However, a straightforward variation to (14) captures the spirit of the strong Wolfe conditions, that large positive derivatives should not be accepted: Assuming $f'(0) < 0$ (i.e. that the search direction is a descent direction), the strong second Wolfe condition can be written exactly as

$$0 \le b_t = f'(t) - c_2 f(0) \le -2c_2 f'(0). \tag{15}$$

The value $-2c_2 f'(0)$ is bounded to 95% confidence by

$$-2c_2 f'(0) \lesssim -2c_2(|\mu'(0)| + 2\sqrt{\mathbb{V}'(0)}) =: \bar{b}. \tag{16}$$

Hence, an approximation to the strong Wolfe conditions can be reached by replacing the infinite upper integration limit on b in Eq. (14) with $(\bar{b} - m_t^b)/\sqrt{C_t^{bb}}$. The effect of this adaptation, which adds no overhead to the computation, is shown in Figure 2 as a dashed line.

3.4 Eliminating Hyper-parameters

As a black-box inner loop, the line search should not require any tuning by the user. The preceding section introduced six so-far undefined parameters: $c_1, c_2, c_W, \theta, \sigma_f, \sigma_{f'}$. We will now show that c_1, c_2, c_W, can be fixed by hard design decisions. θ can be eliminated by standardizing the optimization objective within the line search; and the noise levels can be estimated at runtime with low overhead for batch objectives of the form in Eq. (1). The result is a parameter-free algorithm that effectively *removes* the one most problematic parameter from SGD—the learning rate.

Design Parameters c_1, c_2, c_W Our algorithm inherits the Wolfe thresholds c_1 and c_2 from its deterministic ancestors. We set $c_1 = 0.05$ and $c_2 = 0.8$. This is a standard setting that yields a 'lenient' line search, i.e. one that accepts most descent points. The rationale is that the stochastic aspect of SGD is not always problematic, but can also be helpful through a kind of 'annealing' effect.

The acceptance threshold c_W is a new design parameter arising only in the probabilistic setting. We fix it to $c_W = 0.3$. To motivate this value, first note that in the noise-free limit, all values $0 < c_W < 1$ are equivalent, because p^{Wolfe} then switches discretely between 0 and 1 upon observation of the function. A back-of-the-envelope computation (left out for space), assuming only two evaluations at $t = 0$ and $t = t_1$ and the same fixed noise level on f and f' (which then cancels out), shows that function values barely fulfilling the conditions, i.e. $a_{t_1} = b_{t_1} = 0$, can have $p^{\text{Wolfe}} \sim 0.2$ while function values at $a_{t_1} = b_{t_1} = -\epsilon$ for $\epsilon \to 0$ with 'unlucky' evaluations (both function and gradient values one standard-deviation from true value) can achieve $p^{\text{Wolfe}} \sim 0.4$. The choice $c_W = 0.3$ balances the two competing desiderata for precision and recall. Empirically (Fig. 3), we rarely observed values of p^{Wolfe} close to this threshold. Even at high evaluation noise, a function evaluation typically either clearly rules out the Wolfe conditions, or lifts p^{Wolfe} well above the threshold.

Scale θ The parameter θ of Eq. (4) simply scales the prior variance. It can be eliminated by scaling the optimization objective: We set $\theta = 1$ and scale $y_i \leftarrow (y_i - y_0)/|y_0'|, y_i' \leftarrow y_i'/|y_0'|$ within the code of the line search. This gives $y(0) = 0$ and $y'(0) = -1$, and typically ensures the objective ranges in the single digits across $0 < t < 10$, where most line searches take place. The division by $|y_0'|$ causes a non-Gaussian disturbance, but this does not seem to have notable empirical effect.

Noise Scales $\sigma_f, \sigma_{f'}$ The likelihood (3) requires standard deviations for the noise on both function values (σ_f) and gradients ($\sigma_{f'}$). One could attempt to learn these across several line searches. However, in exchangeable models, as captured by Eq. (1), the variance of the loss and its gradient can be estimated directly within the batch, at low computational overhead—an approach already advocated by Schaul et al. [14]. We collect the empirical statistics

$$\hat{S}(x) := \frac{1}{m} \sum_j^m \ell^2(x, y_j), \quad \text{and} \quad \hat{\nabla}S(x) := \frac{1}{m} \sum_j^m \nabla \ell(x, y_j).^2 \tag{17}$$

6

(where $.^2$ denotes the element-wise square) and estimate, at the beginning of a line search from x_k,

$$\sigma_f^2 = \frac{1}{m-1}\left(\hat{S}(x_k) - \hat{\mathcal{L}}(x_k)^2\right) \qquad \text{and} \qquad \sigma_{f'}^2 = s_i.^{2\top}\left[\frac{1}{m-1}\left(\hat{\nabla}S(x_k) - (\nabla\hat{\mathcal{L}}).^2\right)\right]. \tag{18}$$

This amounts to the cautious assumption that noise on the gradient is independent. We finally scale the two empirical estimates as described in §3.4: $\sigma_f \leftarrow \sigma_f / |y'(0)|$, and ditto for $\sigma_{f'}$. The overhead of this estimation is small if the computation of $\ell(x, y_j)$ itself is more expensive than the summation over j (in the neural network examples of §4, with their comparably simple ℓ, the additional steps added only $\sim 1\%$ cost overhead to the evaluation of the loss). Of course, this approach requires a batch size $m > 1$. For single-sample batches, a running averaging could be used instead (single-sample batches are not necessarily a good choice. In our experiments, for example, vanilla SGD with batch size 10 converged faster in wall-clock time than unit-batch SGD). Estimating noise separately for each input dimension captures the often inhomogeneous structure among gradient elements, and its effect on the noise along the projected direction. For example, in deep models, gradient noise is typically higher on weights between the input and first hidden layer, hence line searches along the corresponding directions are noisier than those along directions affecting higher-level weights.

3.4.1 Propagating Step Sizes Between Line Searches

As will be demonstrated in §4, the line search can find good step sizes even if the length of the direction s_i (which is proportional to the learning rate α in SGD) is mis-scaled. Since such scale issues typically persist over time, it would be wasteful to have the algorithm re-fit a good scale in each line search. Instead, we propagate step lengths from one iteration of the search to another: We set the initial search direction to $s_0 = -\alpha_0 \nabla\hat{\mathcal{L}}(x_0)$ with some initial learning rate α_0. Then, after each line search ending at $x_i = x_{i-1} + t_* s_i$, the next search direction is set to $s_{i+1} = -1.3 \cdot t_* \alpha_0 \nabla\hat{\mathcal{L}}(x_i)$. Thus, the next line search starts its extrapolation at 1.3 times the step size of its predecessor.

Remark on convergence of SGD with line searches: We note in passing that it is straightforward to ensure that SGD instances using the line search inherit the convergence guarantees of SGD: Putting even an extremely loose bound $\bar{\alpha}_i$ on the step sizes taken by the i-th line search, such that $\sum_i^\infty \bar{\alpha}_i = \infty$ and $\sum_i^\infty \bar{\alpha}_i^2 < \infty$, ensures the line search-controlled SGD converges in probability [1].

4 Experiments

Our experiments were performed on the well-worn problems of training a 2-layer neural net with logistic nonlinearity on the MNIST and CIFAR-10 datasets.[6] In both cases, the network had 800 hidden units, giving optimization problems with $636\,010$ and $2\,466\,410$ parameters, respectively. While this may be 'low-dimensional' by contemporary standards, it exhibits the stereotypical challenges of stochastic optimization for machine learning. Since the line search deals with only univariate subproblems, the extrinsic dimensionality of the optimization task is not particularly relevant for an empirical evaluation. Leaving aside the cost of the function evaluations themselves, computation cost associated with the line search is independent of the extrinsic dimensionality.

The central nuisance of SGD is having to choose the learning rate α, and potentially also a schedule for its decrease. Theoretically, a decaying learning rate is necessary to guarantee convergence of SGD [1], but empirically, keeping the rate constant, or only decaying it cautiously, often work better (Fig. 4). In a practical setting, a user would perform exploratory experiments (say, for 10^3 steps), to determine a good learning rate and decay schedule, then run a longer experiment in the best found setting. In our networks, constant learning rates of $\alpha = 0.75$ and $\alpha = 0.08$ for MNIST and CIFAR-10, respectively, achieved the lowest test error after the first 10^3 steps of SGD. We then trained networks with vanilla SGD with and without α-decay (using the schedule $\alpha(i) = \alpha_0/i$), and SGD using the probabilistic line search, with α_0 ranging across five orders of magnitude, on batches of size $m = 10$.

Fig. 4, top, shows test errors after 10 epochs as a function of the initial learning rate α_0 (error bars based on 20 random re-starts). Across the broad range of α_0 values, the line search quickly identified good step sizes $\alpha(t)$, stabilized the training, and progressed efficiently, reaching test errors similar

---

[6] http://yann.lecun.com/exdb/mnist/ and http://www.cs.toronto.edu/~kriz/cifar.html. Like other authors, we only used the "batch 1" sub-set of CIFAR-10.
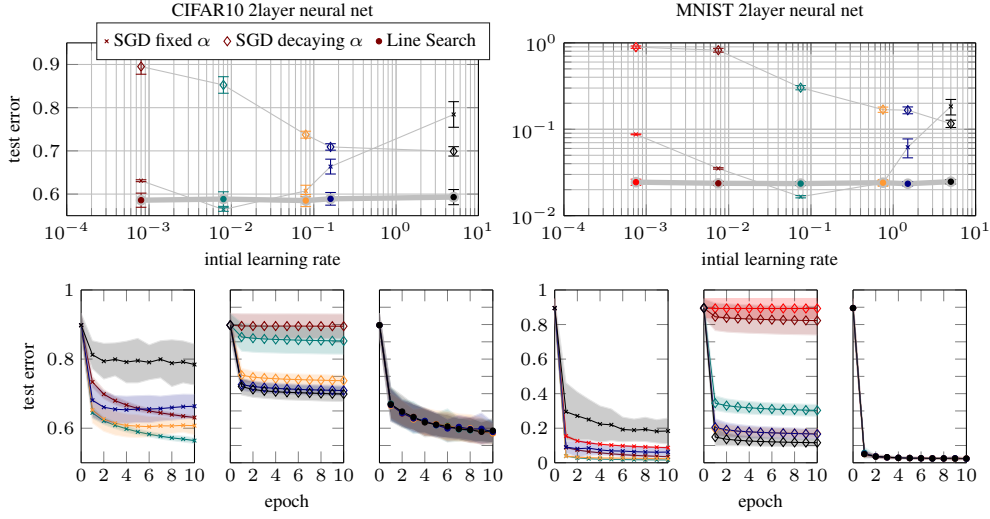
Figure 4: Top row: test error after 10 epochs as function of initial learning rate (note logarithmic ordinate for MNIST). Bottom row: Test error as function of training epoch (same color and symbol scheme as in top row). No matter the initial learning rate, the line search-controlled SGD perform close to the (in practice unknown) optimal SGD instance, effectively removing the need for exploratory experiments and learning-rate tuning. All plots show means and 2 std.-deviations over 20 repetitions.

to those reported in the literature for tuned versions of this kind of architecture on these datasets. While in both datasets, the best SGD instance without rate-decay just barely outperformed the line searches, the optimal α value was *not* the one that performed best after 10^3 steps. So this kind of exploratory experiment (which comes with its own cost of human designer time) would have led to worse performance than simply starting a single instance of SGD with the linesearch and $\alpha_0 = 1$, letting the algorithm do the rest.

Average time overhead (i.e. excluding evaluation-time for the objective) was about 48ms per line search. This is *independent* of the problem dimensionality, and expected to drop significantly with optimized code. Analysing one of the MNIST instances more closely, we found that the average length of a line search was ~ 1.4 function evaluations, $80\% - 90\%$ of line searches terminated after the first evaluation. This suggests good scale adaptation and thus efficient search (note that an 'optimally tuned' algorithm would always lead to accepts).

The supplements provide additional plots, of raw objective values, chosen step-sizes, encountered gradient norms and gradient noises during the optimization, as well test-vs-train error plots, for each of the two datasets, respectively. These provide a richer picture of the step-size control performed by the line search. In particular, they show that the line search chooses step sizes that follow a nontrivial dynamic over time. This is in line with the empirical truism that SGD requires tuning of the step size during its progress, a nuisance taken care of by the line search. Using this structured information for more elaborate analytical purposes, in particular for convergence estimation, is an enticing prospect, but beyond the scope of this paper.

5 Conclusion

The line search paradigm widely accepted in deterministic optimization can be extended to noisy settings. Our design combines existing principles from the noise-free case with ideas from Bayesian optimization, adapted for efficiency. We arrived at a lightweight "black-box" algorithm that exposes no parameters to the user. Our method is complementary to, and can in principle be combined with, virtually all existing methods for stochastic optimization that adapt a step *direction* of fixed *length*. Empirical evaluations suggest the line search effectively frees users from worries about the choice of a learning rate: Any reasonable initial choice will be quickly adapted and lead to close to optimal performance. Our matlab implementation will be made available at time of publication of this article.

References

[1] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, Sep. 1951.

[2] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Twenty-first International Conference on Machine Learning (ICML 2004)*, 2004.

[3] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 19th Int. Conf. on Computational Statistic (COMPSTAT)*, pages 177–186. Springer, 2010.

[4] M.D. Hoffman, D.M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[5] J. Hensman, M. Rattray, and N.D. Lawrence. Fast variational inference in the conjugate exponential family. In *Advances in Neural Information Processing Systems (NIPS 25)*, pages 2888–2896, 2012.

[6] T. Broderick, N. Boyd, A. Wibisono, A.C. Wilson, and M.I. Jordan. Streaming variational Bayes. In *Advances in Neural Information Processing Systems (NIPS 26)*, pages 1727–1735, 2013.

[7] A.P. George and W.B. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, 65(1):167–198, 2006.

[8] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[9] N.N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Ninth International Conference on Artificial Neural Networks (ICANN) 99*, volume 2, pages 569–574, 1999.

[10] S.-I. Amari, H. Park, and K. Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12(6):1399–1409, 2000.

[11] N.L. Roux and A.W. Fitzgibbon. A fast natural Newton method. In *27th International Conference on Machine Learning (ICML)*, pages 623–630, 2010.

[12] R. Rajesh, W. Chong, D. Blei, and E. Xing. An adaptive learning rate for stochastic variational inference. In *30th International Conference on Machine Learning (ICML)*, pages 298–306, 2013.

[13] P. Hennig. Fast Probabilistic Optimization from Noisy Gradients. In *30th International Conference on Machine Learning (ICML)*, 2013.

[14] T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. In *30th International Conference on Machine Learning (ICML-13)*, pages 343–351, 2013.

[15] R. Fletcher and C.M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

[16] C.G. Broyden. A new double-rank minimization algorithm. *Notices of the AMS*, 16:670, 1969.

[17] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317, 1970.

[18] D. Goldfarb. A family of variable metric updates derived by variational means. *Math. Comp.*, 24(109):23–26, 1970.

[19] D.F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Math. Comp.*, 24(111):647–656, 1970.

[20] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Verlag, 1999.

[21] P. Wolfe. Convergence conditions for ascent methods. *SIAM Review*, pages 226–235, 1969.

[22] L. Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

[23] D.R. Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[24] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. MIT, 2006.

[25] G. Wahba. *Spline models for observational data*. Number 59 in CBMS-NSF Regional Conferences series in applied mathematics. SIAM, 1990.

[26] S. Särkkä. *Bayesian filtering and smoothing*. Cambridge University Press, 2013.

[27] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York, 3rd ed. edition, 1991.

[28] R.J. Adler. *The Geometry of Random Fields*. Wiley, 1981.

[29] Z. Drezner and G.O. Wesolowsky. On the computation of the bivariate normal integral. *Journal of Statistical Computation and Simulation*, 35(1-2):101–107, 1990.