Fast Probabilistic Optimization from Noisy Gradients

Philipp Hennig PHILIPP.HENNIG@TUEBINGEN.MPG.DE
Max Planck Institute for Intelligent Systems, Dpt. of Empirical Inference, Spemannstr. Tübingen, Germany

Abstract

Stochastic gradient descent remains popular in large-scale machine learning, on account of its very low computational cost and robustness to noise. However, gradient descent is only linearly efficient and not transformation invariant. Scaling by a local measure can substantially improve its performance. One natural choice of such a scale is the Hessian of the objective function: Were it available, it would turn linearly efficient gradient descent into the quadratically efficient Newton-Raphson optimization. Existing covariant methods, though, are either super-linearly expensive or do not address noise. Generalising recent results, this paper constructs a nonparametric Bayesian quasi-Newton algorithm that learns gradient and Hessian from *noisy* evaluations of the gradient. Importantly, the resulting algorithm, like stochastic gradient descent, has cost linear in the number of input dimensions.

1. Introduction

Much of machine learning involves nonlinear optimization of a negative log likelihood, loss, energy, or other objective function

$$f(\boldsymbol{Z};\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} f(z_i;\boldsymbol{x}) \tag{1}$$

of the data set $\boldsymbol{Z} = \{z_1,\dots,z_N\}$ and the parameters $\boldsymbol{x}$ to be optimized. All efficient optimization algorithms require the gradient $\nabla_{\boldsymbol{x}} f$ of the objective function (shortened to ∇f from here). But if the dataset is very large it can be impossible to evaluate the loss on the entire dataset, and subsets $\boldsymbol{\zeta}_j = \{z_{j_1}, z_{j_2},\dots,z_{j_{N_j}}\} \subset \boldsymbol{Z}$ known as *mini-batches* are used instead.

$$f(\boldsymbol{\zeta}_j;\boldsymbol{x}) = \frac{1}{N_j}\sum_{i=1}^{N_j} f(z_{j_i};\boldsymbol{x}) \tag{2}$$

If the mini-batches are chosen iid from the dataset, the result is a *stochastic gradient* $\nabla f(\zeta_j;\boldsymbol{x}) = \nabla f(\boldsymbol{Z},\boldsymbol{x}) + \xi_j$ whose value corresponds to the true gradient $\nabla f(\boldsymbol{Z},\boldsymbol{x})$ up to some noise ξ_j, which, by the central limit theorem, is approximately Gaussian distributed. A typical example of this setup is the training of neural networks by back-propagation (Rumelhart et al., 1986).

The most straightforward, but not the most efficient use of the gradient for optimization is *gradient descent*—iteratively updating the parameters according to $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha_t \nabla f(\boldsymbol{Z};\boldsymbol{x}_t)$, or, in the mini-batch version, *stochastic gradient descent*, $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha_t \nabla f(\boldsymbol{z}_i;\boldsymbol{x}_t)$. In each case, $\alpha_t \in \mathbb{R}_+$ is a learning rate. There is some basic theory on setting α (e.g. Robbins & Monro, 1951). For example, choosing $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$ guarantees convergence in the limit $t \to \infty$. But it does not yield efficiency. In fact, it is clear from inspection that the gradient descent learning rule is not invariant under even linear changes of measure: The units of measure of ∇f are those of f/x, not those of x. If the task were to minimize potential energy density by skiing down a slope, a European gradient descent algorithm ($\nabla f = 5\mathrm{J}/[\mathrm{kg}\cdot\mathrm{m}]$) would move in increments of metres, while it's American cousin would be much more cautious on the same slope ($\nabla f \approx 10^{-5}\mathrm{Cal}/[\mathrm{oz}\cdot\mathrm{ft}]$), moving only in increments of ten-thousandths of a foot (3µm). Simple standardisation does not solve this fundamental problem: what is a "standard" gradient at initiation may turn out to be a steep or flat gradient in other regions. In particular, gradient descent can be very slow in plateau regions, and stochastic gradient descent can effectively get stuck in diffusion in such areas. The long established solution to this problem is to correct the gradient descent direction relative to a local measure of f of the right units $[f/x^2]$. A classic, though not unique, choice is the Newton-Raphson algorithm, which corrects by the local curvature, the

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

Hessian $B(\boldsymbol{x}) = \nabla\nabla^\intercal f(\boldsymbol{Z};\boldsymbol{x})$:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - B^{-1}(\boldsymbol{x}_t)\nabla f(\boldsymbol{Z};\boldsymbol{x}_t). \tag{3}$$

This algorithm is often described as *efficient*, both in the sense that, if L were a quadratic, it would reach the minimum in one step, and in the sense that there is a neighbourhood around each local minimum $\boldsymbol{x}^*$ such that, if the algorithm reaches this neighbourhood, the estimate $\boldsymbol{x}_t$ converges to $\boldsymbol{x}^*$ quadratically fast (Dennis & Morée, 1977) – although this neighbourhood can be arbitrarily small. Unfortunately, Newton's algorithm is too expensive for high-dimensional problems: If $\boldsymbol{x} \in \mathbb{R}^N$, constructing the Hessian has general cost $\mathcal{O}(N^2)$, its inversion $\mathcal{O}(N^3)$. An elegant way to address this issue is offered by *quasi-Newton* algorithms (Fletcher & Powell, 1963; Broyden, 1965), which are learning methods constructing low-rank estimators for B^{-1} from observations of only the gradient. A particularly cost-efficient member of this class is the L-BFGS algorithm (Nocedal, 1980; Broyden, 1969; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970), which can construct such an estimator in $\mathcal{O}(N)$, linear in the input dimensionality. Unfortunately, though, quasi-Newton methods do not apply to noisy gradient observations. Several authors have proposed computationally elegant ways of re-scaling the gradient using the Hessian (Pearlmutter, 1994; Schraudolph, 2002; Martens, 2010; Martens & Sutskever, 2011), sometimes leveraging the structure (e.g. sparsity) of specific optimization problems (Bordes et al., 2009). But, again, these "Hessian-free" approaches do not properly account for noise. Their cost is also not usually $\mathcal{O}(N)$ in a strict sense, or they involve sampling (Byrd et al., 2011).

Quasi-Newton methods are touchstones of numerical analysis. Decades after their invention, they remain state of the art in nonlinear optimization. Recently, work by Hennig & Kiefel (2012) has provided novel, unifying insight into this family of methods, casting them as an approximation to Bayesian linear regression. The cited work established that the low computational cost of algorithms like BFGS is achieved by deliberately ignoring or weakening prior knowledge, such as the symmetry of the Hessian, whose exact encoding would have at least cubic cost. These authors then focussed on extending the classic algorithms to a nonparametric paradigm, which increases performance in various ways. This paper generalises these results to noisy observations. This is nontrivial, because the noisy setting requires a more detailed study of the information content of each observation. As Hennig & Kiefel showed, classic quasi-Newton methods can be interpreted as using each observation twice, in two independent likelihood terms. When considering noisy observations, we thus have to make sure that these observations do not amount to double counting. This paper shows that quasi-Newton methods are in fact *more* cautious than necessary. The guiding insight is that they *deliberately ignore structural knowledge*, in exchange for low computational cost, to a surprisingly extreme extent: the algorithms ignore the isomorphism between vectors and co-vectors. Where much of machine learning is about using structure to increase learning efficiency, here the approach is to remove structure to decrease cost. The final result is a nonparametric Bayesian quasi-Newton algorithm for noisy observations. Its most important feature is that it is $\mathcal{O}(N)$, linear in the number of parameters to optimize. So, although the new algorithm is more expensive than gradient descent by a considerable factor, its cost *scales* like that of gradient descent. The new algorithm can thus be applied to very high-dimensional optimization problems, including the training of neural networks (Section 3.2).

2. Methods

Since Hennig & Kiefel (2012) focussed on unifying a group of classic algorithms, their nonparametric derivation started from a parametric viewpoint. Here, a nonparametric model is derived from scratch, which allows a more direct exposition. The following section constructs related priors over f, ∇f, and the Hessian B. The model is interesting less for what it encodes than for what it does *not* encode.

2.1. Prior

The objective function f is defined over an input space $\mathbb{R}^N$. This could be the vector space $\mathbb{R}^{N\times 1}$ of N-dimensional real vectors, or just as well its *dual vector space* $\mathbb{R}^{1\times N}$, the space of linear maps from $\mathbb{R}^{N\times 1}$ to $\mathbb{R}$. This distinction is usually irrelevant, because these spaces are isomorphic: There is a bijection between $x \in \mathbb{R}^{N\times 1}$ and $x^\intercal \in \mathbb{R}^{1\times N}$—transposition—preserving the vector space structure. The following derivations *deliberately ignore* this structural knowledge, and define the objective function over a $2N$ dimensional vector space $\mathbb{R}^{1\times N} \times \mathbb{R}^{N\times 1}$ (isomorphic to $\mathbb{R}^{2N}$). To avoid accidentally using the isomorphism between vectors and co-vectors, I introduce the new notation $\boldsymbol{x}^\intercal \equiv \boldsymbol{x}_\triangleleft$ and $\boldsymbol{x} \equiv \boldsymbol{x}_\triangleright$. Thence, consider a Gaussian process prior over f

$$p(f(\boldsymbol{x}_\triangleleft, \boldsymbol{x}_\triangleright)) = \mathcal{GP}(f; \mu_f, k) \quad \text{with} \tag{4}$$

$$k([\boldsymbol{x}^\triangledown_\triangleleft, \boldsymbol{x}^\triangledown_\triangleright], [\boldsymbol{x}^\vartriangle_\triangleleft, \boldsymbol{x}^\vartriangle_\triangleright]) \equiv k^\triangleleft_{\boldsymbol{x}^\triangledown_\triangleleft \boldsymbol{x}^\vartriangle_\triangleleft} \cdot k^\triangleright_{\boldsymbol{x}^\triangledown_\triangleright \boldsymbol{x}^\vartriangle_\triangleright}, \tag{5}$$

with mean function $\mu_f : \mathbb{R}^{1\times N} \times \mathbb{R}^{N\times 1} \to \mathbb{R}$ and covariance functions (kernels) $k^\triangleleft : \mathbb{R}^{1\times N} \times \mathbb{R}^{1\times N} \to \mathbb{R}$;

$k^{\triangleright} : \mathbb{R}^{N\times 1} \times \mathbb{R}^{N\times 1} \to \mathbb{R}$. The notation $x^{\blacktriangle}$ and $x^{\blacktriangledown}$ (read "x up" and "x down") denotes two arbitrary separate N-dimensional inputs. It is a variant of the more commonly used notation x_* and x^*, which will not be used here to prevent confusion over the use of sub- and superscripts. The product form of the covariance assumes independent behaviour of the function in the two N-dimensional sub-spaces. It is thus strictly more conservative than a classic N-dimensional prior (which implicitly assumes that $\boldsymbol{x}_{\triangleleft}$ and $\boldsymbol{x}_{\triangleright}$ are the same thing, so the function values must be perfectly correlated between those two sub-spaces). For a concrete implementation, consider the popular squared exponential (SE) kernels, which contain a further independence assumption about the function's form in each of the input dimensions:

$$k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}\boldsymbol{x}^{\blacktriangle}_{\triangleleft}} = \theta^2_{\triangleleft} \exp\left[-\frac{1}{2}\sum_n^N \frac{(\boldsymbol{x}^{\blacktriangledown}_{\triangleleft} - \boldsymbol{x}^{\blacktriangle}_{\triangleleft})^2_n}{\lambda^2_{\triangleleft n}}\right] \tag{6}$$

$$k^{\triangleright}_{\boldsymbol{x}^{\blacktriangledown}_{\triangleright}\boldsymbol{x}^{\blacktriangle}_{\triangleright}} = \theta^2_{\triangleright} \exp\left[-\frac{1}{2}\sum_n^N \frac{(\boldsymbol{x}^{\blacktriangledown}_{\triangleright} - \boldsymbol{x}^{\blacktriangle}_{\triangleright})^2_n}{\lambda^2_{\triangleright n}}\right]. \tag{7}$$

A weak encoding of the relation between $\boldsymbol{x}_{\triangleleft}$ and $\boldsymbol{x}_{\triangleright}$ can be achieved by setting $\boldsymbol{\lambda}_{\triangleleft} = \boldsymbol{\lambda}_{\triangleright} \equiv \boldsymbol{\lambda}$ and $\theta_{\triangleleft} = \theta_{\triangleright} \equiv \theta$.

Since we have artificially separated vectors and covectors, there are now actually *two* gradients: $\nabla_{\triangleright} f$ and $\nabla_{\triangleleft} f$, the vectors of derivatives of f with respect to the elements of $\boldsymbol{x}_{\triangleright}$ and $\boldsymbol{x}_{\triangleleft}$, respectively. Of course, these are identical up to transposition; but the model does not encode this knowledge. The following derivation is symmetric under transposition, so it suffices to consider only one case. The Gaussian family is closed under linear maps, and differentiation is linear. So the prior over $\nabla_{\triangleright} f$ is also a Gaussian process (Rasmussen & Williams, 2006, §9.4), with mean function $\boldsymbol{\mu}_{\triangleright}(\boldsymbol{x}_{\triangleleft}, \boldsymbol{x}_{\triangleright}) = \nabla_{\triangleright}\mu(\boldsymbol{x}_{\triangleleft}, \boldsymbol{x}_{\triangleright})$, and covariance function (δ_{ab} is Kronecker's symbol, the second line uses the explicit SE kernel of Eq. (6))

$$\begin{aligned}\operatorname{cov}\left[\frac{\partial f(\boldsymbol{x}^{\blacktriangle}_{\triangleleft}, \boldsymbol{x}^{\blacktriangle}_{\triangleright})}{\partial x^{\blacktriangle}_{\triangleright,j}}, \frac{\partial f(\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}, \boldsymbol{x}^{\blacktriangledown}_{\triangleright})}{\partial x^{\blacktriangledown}_{\triangleright,\ell}}\right] &= k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangle}_{\triangleleft}\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}} \frac{\partial^2 k^{\triangleright}_{\boldsymbol{x}^{\blacktriangle}_{\triangleright}\boldsymbol{x}^{\blacktriangledown}_{\triangleright}}}{\partial x^{\blacktriangle}_{\triangleright,j}\partial x^{\blacktriangledown}_{\triangleright,\ell}} \\ = k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangle}_{\triangleleft}\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}} k^{\triangleright}_{\boldsymbol{x}^{\blacktriangle}_{\triangleright}\boldsymbol{x}^{\blacktriangledown}_{\triangleright}} &\left[\frac{\delta_{j\ell}}{\lambda^2_j} + \frac{(\boldsymbol{x}^{\blacktriangledown}_{\triangleright} - \boldsymbol{x}^{\blacktriangle}_{\triangleright})_j (\boldsymbol{x}^{\blacktriangledown}_{\triangleright} - \boldsymbol{x}^{\blacktriangle}_{\triangleright})_\ell}{\lambda^2_j \lambda^2_\ell}\right]. \end{aligned} \tag{8}$$

Unfortunately, due to the second term in the sum, evaluating this covariance has cost $\mathcal{O}(N^2)$. Dropping that term again *deliberately ignores* co-variance structure. The term is strictly positive definite—it is the product of a linear kernel with a SE kernel and a number of positive scalars, thus itself a positive definite kernel, because kernels form a semi-ring (Rasmussen & Williams, 2006, §4.2.4). Since its value at $\boldsymbol{x}^{\blacktriangle}_{\triangleright} = \boldsymbol{x}^{\blacktriangledown}_{\triangleright}$ is zero, it contributes no marginal variance, only covariance. Thus, the resulting Gaussian process prior over $\nabla_{\triangleright} f$ with covariance function

$$\begin{aligned} k^{\nabla}_{j\ell}([\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}, \boldsymbol{x}^{\blacktriangle}_{\triangleright}], [\boldsymbol{x}^{\blacktriangle}_{\triangleleft}, \boldsymbol{x}^{\blacktriangle}_{\triangleright}]) &= \frac{\delta_{j\ell}}{\lambda^2_j} k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangle}_{\triangleleft}\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}} k^{\triangleright}_{\boldsymbol{x}^{\blacktriangle}_{\triangleright}\boldsymbol{x}^{\blacktriangledown}_{\triangleright}} \\ &= \left[k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangle}_{\triangleleft}\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}} \otimes \Lambda k^{\triangleright}_{\boldsymbol{x}^{\blacktriangle}_{\triangleright}\boldsymbol{x}^{\blacktriangledown}_{\triangleright}}\right]_{(1j)(1\ell)} \end{aligned} \tag{9}$$

(with $\Lambda \equiv \operatorname{diag}(\boldsymbol{\lambda}^{-2})$) makes strictly weaker assumptions than Eq. (8), in the sense that it assumes strictly lower covariance between function values at differing points. The second equality introduces the Kronecker product between matrices, $(a \otimes b)_{(ij)(k\ell)} = a_{ik} b_{j\ell}$, which will appear repeatedly from here on. It is particularly useful when dealing with operator-valued functions: If we stack the elements of matrix B_{ij}, row by row, into the vector $\overrightarrow{B}_{(ij)}$, then matrix products can be represented as $\overrightarrow{ABC^{\intercal}} = (A \otimes C)\overrightarrow{B}$. Since the chosen kernel on f is symmetric under transposition, the GP prior on $\nabla_{\triangleleft}$ has kernel $\Lambda k^{\triangleleft} \otimes k^{\triangleright}$, identical to Eq. (9) up to transposition.

The elements of the Hessian $B(\boldsymbol{x}_{\triangleleft}, \boldsymbol{x}_{\triangleright}) : \mathbb{R}^{1\times N} \times \mathbb{R}^{N\times 1} \to \mathbb{R}$ are $B_{ij} = \partial_{x_{\triangleleft},i}\partial_{x_{\triangleright},j} f(\boldsymbol{x}_{\triangleleft}, \boldsymbol{x}_{\triangleright})$, the derivatives of the elements of $\nabla_{\triangleright}$ with respect to the elements of $\boldsymbol{x}_{\triangleleft}$. But they are *also* $\partial_{x_{\triangleright},i}\partial_{x_{\triangleleft},j} f(\boldsymbol{x}_{\triangleleft}, \boldsymbol{x}_{\triangleright})$, the derivatives of $\nabla_{\triangleleft}$ with respect to the elements of $\boldsymbol{x}_{\triangleright}$. Under the factorisation structure of the prior, these two facts are now separate, *independent*, statements. Using the first definition, and the same argument as for Eq. (8), the belief over these elements is also a Gaussian process, with mean $B_0(\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}, \boldsymbol{x}^{\blacktriangledown}_{\triangleright}) = \partial_{\boldsymbol{x}_{\triangleleft}} \boldsymbol{\mu}_{\triangleright}$ and covariance

$$\operatorname{cov}\left[\frac{\partial^2 f(\boldsymbol{x}^{\blacktriangle}_{\triangleleft}, \boldsymbol{x}^{\blacktriangle}_{\triangleright})}{\partial x^{\blacktriangle}_{\triangleleft,i}\partial x^{\blacktriangle}_{\triangleright,j}}, \frac{\partial^2 f(\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}, \boldsymbol{x}^{\blacktriangledown}_{\triangleright})}{\partial x^{\blacktriangledown}_{\triangleleft,k}\partial x^{\blacktriangledown}_{\triangleright,\ell}}\right] \tag{10}$$

$$= \frac{\partial^2 k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangle}_{\triangleleft}\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}}}{\partial x^{\blacktriangle}_{\triangleleft,i}\partial x^{\blacktriangledown}_{\triangleleft,k}} \frac{\partial^2 k^{\triangleright}_{\boldsymbol{x}^{\blacktriangle}_{\triangleright}\boldsymbol{x}^{\blacktriangledown}_{\triangleright}}}{\partial x^{\blacktriangle}_{\triangleright,j}\partial x^{\blacktriangledown}_{\triangleright,\ell}} \tag{11}$$

$$= k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangle}_{\triangleleft}\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}} k^{\triangleright}_{\boldsymbol{x}^{\blacktriangle}_{\triangleright}\boldsymbol{x}^{\blacktriangledown}_{\triangleright}} \left[\frac{\delta_{j\ell}}{\lambda^2_j} + \frac{(\boldsymbol{x}^{\blacktriangledown}_{\triangleright} - \boldsymbol{x}^{\blacktriangle}_{\triangleright})_j (\boldsymbol{x}^{\blacktriangledown}_{\triangleright} - \boldsymbol{x}^{\blacktriangle}_{\triangleright})_\ell}{\lambda^2_j \lambda^2_\ell}\right] \cdot \left[\frac{\delta_{ik}}{\lambda^2_i} + \frac{(\boldsymbol{x}^{\blacktriangledown}_{\triangleright} - \boldsymbol{x}^{\blacktriangle}_{\triangleright})_i (\boldsymbol{x}^{\blacktriangledown}_{\triangleright} - \boldsymbol{x}^{\blacktriangle}_{\triangleright})_k}{\lambda^2_i \lambda^2_k}\right]. \tag{12}$$

Just as above we drop the quadratically expensive mixing terms, to get the strictly weaker kernel

$$\begin{aligned} k^{B}_{(ij)(k\ell)}([\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}, \boldsymbol{x}^{\blacktriangle}_{\triangleright}], [\boldsymbol{x}^{\blacktriangle}_{\triangleleft}, \boldsymbol{x}^{\blacktriangle}_{\triangleright}]) &= \frac{\delta_{j\ell}\delta_{ik}}{\lambda^2_i \lambda^2_j} k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangle}_{\triangleleft}\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}} k^{\triangleright}_{\boldsymbol{x}^{\blacktriangle}_{\triangleright}\boldsymbol{x}^{\blacktriangledown}_{\triangleright}} \\ &= \left[\Lambda k^{\triangleleft}_{\boldsymbol{x}^{\blacktriangle}_{\triangleleft}\boldsymbol{x}^{\blacktriangledown}_{\triangleleft}} \otimes \Lambda k^{\triangleright}_{\boldsymbol{x}^{\blacktriangle}_{\triangleright}\boldsymbol{x}^{\blacktriangledown}_{\triangleright}}\right]_{(ij)(k\ell)}. \end{aligned} \tag{13}$$

An important observation is that the second definition of B (that is, reversing the order of differentiation),

yields the *same* kernel k^B for the Hessian. This is not surprising—it is the reason why the Hessian is symmetric—but will become relevant in Section 2.2.

Together, the derivations so far provide independent Gaussian process priors for f, the $2N$ gradient elements, and the N^2 elements of the Hessian, each as functions over the $2N$-dimensional input space. Dropping correlation between pairwise different dimensions also weakens the relation between the belief over the f and its gradients. But the derivation above does retain a relation between the length scales $\boldsymbol{\lambda}$ and the signal variances of gradient and Hessian: If f has large length scales, it *changes* slowly, so we expect smaller *values* for gradient and Hessian.

If we had not split $\boldsymbol{x}_\triangleright$ and $\boldsymbol{x}_\triangleleft^\top$, the dropped terms would have encoded the fact that the Hessian is symmetric, and that gradient and Hessian are conservative fields (i.e. that line integrals over them are independent of the integration path). These aspects are now erased from the model, so the posterior will converge more conservatively than it otherwise would. In return for this weakened inference power, it will turn out that the resulting inference algorithm is only linearly expensive.

2.2. Inference

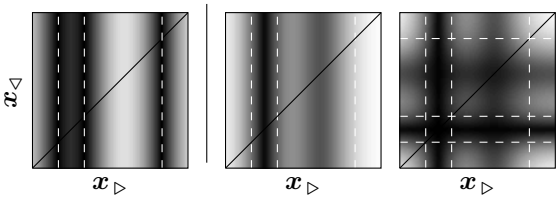

Figure 1. Conceptual sketches, showing posterior *marginal variances (uncertainties)*, not mean functions, after three observations at locations marked by white dashed lines. **Left:** posterior over a gradient element. **Middle:** posterior over an element of the Hessian's *main diagonal* after the "primal" observation (Eq. 22). **Right:** posterior over the same Hessian element after both "primal" and "dual" observation (Eq. 28). All plots use the same color scale, from 0 (black) to unit (white) uncertainty. The dual observation *does* contain additional information, even along the $\boldsymbol{x}_\triangleleft = \boldsymbol{x}_\triangleright$ sub-space.

Estimating the Newton descent direction $-B^{-1}\nabla f$ requires beliefs for both gradient and Hessian, from observations of only the gradients, constructed these in this section. Figure 1 gives intuition.

For the purposes of this paper, we will assume independent Gaussian noise of standard deviation α on all elements of the gradient. In applications like the training of neural networks, the strength of the noise may well depend on the function value. In such cases simple heuristics may help improve performance.

2.2.1. Inference on the gradient

Observing $\nabla_\triangleright f$ means observing the derivative of f *along the entire subspace spanned by* $\boldsymbol{x}_\triangleleft$. We thus observe noisy *functions* (not just single function values) $\boldsymbol{y}_m(\boldsymbol{x}_\triangleleft, \boldsymbol{x}_\triangleright^m)$, $m = 0, \dots, M$, which are constant along $\boldsymbol{x}_\triangleleft$. The likelihood is thus $p(\boldsymbol{y}_m(\boldsymbol{x}_\triangleleft) \,|\, \boldsymbol{x}_\triangleright^m, \nabla_\triangleright f) = \mathcal{N}(\boldsymbol{y}_m(\boldsymbol{x}_\triangleleft); \nabla_\triangleright f(\boldsymbol{x}_\triangleleft, \boldsymbol{x}_\triangleright^m), k^\triangleleft \cdot \alpha^2 \boldsymbol{I})$. The presence of a kernel function in this likelihood may initially be confusing, but it is simply the correct encoding of an observation containing no information in the $\boldsymbol{x}_\triangleleft$ space.

The joint conditional probability of the $M+1$ vectors $\boldsymbol{y}_m$, observed at $M+1$ locations $\boldsymbol{x}_m$, combined into the $N \times M+1$ matrices $\boldsymbol{Y}$ and $\boldsymbol{X}$, respectively, is

$$p(\boldsymbol{Y} \,|\, \boldsymbol{X}, \nabla f) = \mathcal{N}(\overrightarrow{\boldsymbol{Y}}; \overrightarrow{\nabla \boldsymbol{f}}(\boldsymbol{X}_\triangleleft, \boldsymbol{X}_\triangleright), k^\triangleleft \otimes \boldsymbol{I}_N \otimes \alpha^2 \boldsymbol{I}_{M+1}) \tag{14}$$

Inference is analytic: The posterior over the gradient is a Gaussian process with mean function

$$\begin{aligned} \boldsymbol{\mu}_\nabla^{\boldsymbol{Y}}(\boldsymbol{x}_\triangleleft^\blacktriangledown, \boldsymbol{x}_\triangleright^\blacktriangledown) &= \boldsymbol{\mu}_\nabla(\boldsymbol{x}_\triangleleft^\blacktriangledown, \boldsymbol{x}_\triangleright^\blacktriangledown) + (k^\triangleleft \otimes \Lambda \boldsymbol{k}^\triangleright_{\boldsymbol{x}^\blacktriangledown \boldsymbol{X}}) \cdot \qquad (15) \\ &\left[k^\triangleleft \otimes (\Lambda \boldsymbol{k}^\triangleright_{\boldsymbol{X},\boldsymbol{X}} + \alpha \boldsymbol{I}_{N \otimes M})\right]^{-1} (\overrightarrow{\boldsymbol{Y}}(x_\triangleleft^\blacktriangledown) - \overrightarrow{\boldsymbol{\mu}}_\nabla(\boldsymbol{x}_\triangleleft^\blacktriangledown, \boldsymbol{X}_\triangleright)) \\ &= \boldsymbol{\mu}_\nabla(\boldsymbol{x}_\triangleleft^\blacktriangledown, \boldsymbol{x}_\triangleright^\blacktriangledown) + \Lambda \boldsymbol{k}^\triangleright_{\boldsymbol{x}^\blacktriangledown \boldsymbol{X}} (\Lambda \boldsymbol{k}^\triangleright_{\boldsymbol{X},\boldsymbol{X}} + \alpha \boldsymbol{I}_{N \otimes M})^{-1} \cdot \qquad (16) \\ &\quad (\overrightarrow{\boldsymbol{Y}} - \overrightarrow{\boldsymbol{\mu}}_\nabla(\boldsymbol{x}_\triangleleft^\blacktriangledown, \boldsymbol{X}_\triangleright)). \end{aligned}$$

Assuming independence between elements (Λ is diagonal), this expression simplifies into separate expressions for each dimension $n \in \{1, \dots, N\}$. Writing $\boldsymbol{Y}_{n:}$ for the n-th row of $\boldsymbol{Y}$, we find

$$\begin{aligned} \mu^{\boldsymbol{Y}_{n:}}_{\nabla,n}(\boldsymbol{x}_\triangleleft^\blacktriangledown, \boldsymbol{x}_\triangleright^\blacktriangledown) &= \mu_{\nabla,n}(\boldsymbol{x}_\triangleleft^\blacktriangledown, \boldsymbol{x}_\triangleright^\blacktriangledown) + \lambda_n^{-2} \boldsymbol{k}^\triangleright_{\boldsymbol{x}^\blacktriangledown, \boldsymbol{X}} \cdot \\ &\quad (\lambda_n^{-2} \boldsymbol{k}^\triangleright_{\boldsymbol{X}\boldsymbol{X}} + \alpha \boldsymbol{I}_M)^{-1} (\boldsymbol{Y}_{n:}^\top - \boldsymbol{\mu}_{\nabla,n}(\boldsymbol{x}_l^\triangleright, \boldsymbol{X}_\triangleright)). \end{aligned} \tag{17}$$

Evaluating this mean function once for all dimensions has cost $\mathcal{O}(NM + M^3 N)$. If all λ_n are identical, which is likely to be a good assumption for many applications, including neural networks, the cost is $\mathcal{O}(NM + M^3)$. The posterior covariance will not feature in the algorithm; but it is also analytic, and independent over the N dimensions:

$$\begin{aligned} &\Sigma_{\nabla,n}(\boldsymbol{x}_\triangleleft^\blacktriangledown, \boldsymbol{x}_\triangleright^\blacktriangledown, \boldsymbol{x}_\triangleleft^\blacktriangle, \boldsymbol{x}_\triangleright^\blacktriangle) = \boldsymbol{k}^\triangleleft_{\boldsymbol{x}^\blacktriangle \boldsymbol{x}^\blacktriangledown} \cdot \\ &\lambda_n^{-2} \left(\boldsymbol{k}^\triangleright_{\boldsymbol{x}^\blacktriangle \boldsymbol{x}^\blacktriangledown} - \lambda_n^{-2} \boldsymbol{k}^\triangleright_{\boldsymbol{x}^\blacktriangle \boldsymbol{X}} (\lambda_n^{-2} \boldsymbol{k}^\triangleright_{\boldsymbol{X}\boldsymbol{X}} + \alpha \boldsymbol{I}_M)^{-1} \boldsymbol{k}^\triangleright_{\boldsymbol{X}\boldsymbol{x}^\blacktriangledown} \right). \end{aligned} \tag{18}$$

2.2.2. Inference on the Hessian

The likelihood of the Hessian is somewhat more involved, but still allows analytic inference. The gradient

is the integral over the Hessian, and integration is a linear operation. Recall that there are two equivalent, but in our model independent, ways of constructing the Hessian. To incorporate both, as in Hennig & Kiefel (2012), and as in classic quasi-Newton methods, we construct a rank-$2M$ update by considering two independent observations, $\boldsymbol{u}^i_\triangleright = \boldsymbol{y}_i - \boldsymbol{y}_{i-1}$ and $\boldsymbol{u}^i_\triangleleft = \boldsymbol{y}^\top_i - \boldsymbol{y}^\top_{i-1}$, $i = 1, \ldots, I$, encoding the integrals over the Hessian along a linear path. We will combine them into an $N \times M$ matrix $\boldsymbol{U}_\triangleright$ and a $M \times N$ matrix $\boldsymbol{U}_\triangleleft$. Since the noise on each $\boldsymbol{y}_i$ is presumed i.i.d. and of variance σ^2, $\boldsymbol{u}_\triangleright$ is the difference of two Gaussian variables, thus has noise variance $2\sigma^2$. We will also use the matrix $\boldsymbol{S}$ with elements $\boldsymbol{S}_{im} \equiv \boldsymbol{x}^m_i - \boldsymbol{x}^{m-1}_i$. The conditional probability of $\boldsymbol{U}_\triangleright$ can then be written as

$$p(\boldsymbol{U}_\triangleright; B, \boldsymbol{S}) = \mathcal{N}\left(\boldsymbol{U}_\triangleright; \mathfrak{S}^\top_\triangleright B, \Lambda k_\triangleleft \otimes 2\sigma^2 \boldsymbol{I}_M\right) \quad (19)$$

This uses the linear operator $\mathfrak{S}_\triangleright = (\boldsymbol{I} \otimes (\int \odot \boldsymbol{S}))$ (where $\odot$ is the Hadamard, or point-wise product). It returns the integral of B along the $\boldsymbol{x}_\triangleright$ sub-space

$$(\mathfrak{S}_\triangleright B)_{im} = \sum_{n=1}^{N} \boldsymbol{S}_{nm} \int_0^1 B_{in}(\boldsymbol{x}_\triangleleft, \boldsymbol{x}_\triangleright(\tau))\, \mathrm{d}\tau \quad \text{with} \quad \boldsymbol{x}_\triangleright(\tau) = (\boldsymbol{x}^m - \boldsymbol{x}^{m-1})\tau + \boldsymbol{x}^{m-1}. \quad (20)$$

The posterior after this first observation is also a Gaussian process. Constructing it is tedious but, given the insights developed in the previous sections, it is now a relatively uncomplicated generalisation of the results of Hennig & Kiefel (2012), where some details can be found. The posterior mean $B_\triangleright$ and covariance function $\Sigma_\triangleright$ over B are

$$B_\triangleright(\boldsymbol{x}^\forall_\triangleleft, \boldsymbol{x}^\forall_\triangleright) = B_0 + (\boldsymbol{U}_\triangleright - \mathfrak{B}_\triangleright)(\mathfrak{K}^\triangleright + 2\alpha^2 \boldsymbol{I}_M)^{-1} \mathfrak{k}^\top_\triangleright(\boldsymbol{x}^\forall_\triangleright), \quad (21)$$

$$\Sigma_\triangleright(\boldsymbol{x}^\forall_\triangleleft, \boldsymbol{x}^\forall_\triangleright, \boldsymbol{x}^\wedge_\triangleleft, \boldsymbol{x}^\wedge_\triangleright) = \Lambda k^\triangleleft \otimes \left[\Lambda k^\triangleright - \mathfrak{k}_\triangleright \left[\mathfrak{K}^\triangleright + 2\alpha^2 \boldsymbol{I}_M\right]^{-1} \mathfrak{k}^\top_\triangleright\right], \quad (22)$$

with the maps $\mathfrak{B}_\triangleright \in \{\mathbb{R}^{1\times N} \to \mathbb{R}^{N\times M}\}$, $\mathfrak{k}^\triangleright \in \{\mathbb{R}^{N\times 1} \to \mathbb{R}^{N\times M}\}$, and the matrix $\mathfrak{K}_\triangleright \in \mathbb{R}^{M\times M}$

$$\mathfrak{B}_{\triangleright,nm}(\boldsymbol{x}^\forall_\triangleleft) \equiv \mathfrak{S}^\top B = \sum_{\ell=1}^{N} S_{\ell m} \int_0^1 B_{0,n\ell}(\boldsymbol{x}^\forall_\triangleleft, \boldsymbol{x}_\ell(\tau))\, \mathrm{d}\tau \quad (23)$$

$$\mathfrak{k}_{\triangleright,nm}(\boldsymbol{x}^\forall_\triangleright) \equiv \mathfrak{S}^\top k^\triangleright = S_{nm}\lambda_n^{-2} \int_0^1 k^\triangleright(\boldsymbol{x}^\forall_\triangleleft, \boldsymbol{x}_n(\tau))\, \mathrm{d}\tau \quad (24)$$

$$\mathfrak{K}^\triangleright_{m\ell} \equiv \mathfrak{S}^\top k^\triangleright \mathfrak{S} = \sum_{n=1}^{N} {\lambda_n}^{-2} S_{nm} S_{n\ell} \iint_0^1 k^\triangleright(\boldsymbol{x}_m(\tau), \boldsymbol{x}_\ell(\tau'))\, \mathrm{d}\tau\, \mathrm{d}\tau'. \quad (25)$$

Assuming the prior mean function B_0 is a constant diagonal matrix, evaluating these two expressions is $\mathcal{O}(N \cdot M^2)$. For the squared exponential kernel, Equations (24) & (25) require definite univariate and bivariate Gaussian integrals. There are no analytic expressions for these integrals; but good (single precision), lightweight numerical approximations exist (Genz, 2004; Hennig & Kiefel, 2012). The second, independent observation in the dual subspace is $\boldsymbol{U}_\triangleleft$, the corresponding likelihood is, using $\mathfrak{S}_\triangleleft = ((\int \odot \boldsymbol{S}) \otimes \boldsymbol{I})$,

$$p(\boldsymbol{U}_\triangleleft; B, \boldsymbol{S}) = \mathcal{N}\big(\boldsymbol{u}_\triangleleft; \mathfrak{S}^\top_\triangleleft B, 2\sigma^2 \boldsymbol{I}_M \otimes \Lambda k^\triangleright - \mathfrak{k}_\triangleright \left[\mathfrak{K}^\triangleright + 2\alpha^2 \boldsymbol{I}_M\right]^{-1} \mathfrak{k}^\top_\triangleright\big). \quad (26)$$

Once again, the presence of the kernel form in this likelihood is simply encoding ignorance in this observation along the orthogonal subspace. By extension of the derivation for Equations (21) & (22) (see also Hennig & Kiefel, 2012), the posterior after *both* independent observations is a Gaussian process with mean function $B_\diamond$ and (here unused) covariance $\Sigma_\diamond$,

$$\begin{aligned} B_\diamond = B_0 &+ (\boldsymbol{U}_\triangleright - \mathfrak{B}_\triangleright)(\mathfrak{K}^\triangleright + 2\alpha^2 \boldsymbol{I}_M)^{-1} \mathfrak{k}^\top_\triangleright(\boldsymbol{x}^\forall_\triangleright) \\ &+ \mathfrak{k}_\triangleleft(\boldsymbol{x}^\forall_\triangleleft)(\mathfrak{K}^\triangleleft + 2\alpha^2 \boldsymbol{I}_M)^{-1}(\boldsymbol{U}_\triangleleft - \mathfrak{B}_\triangleleft)^\top \\ &- \mathfrak{k}_\triangleleft(\boldsymbol{x}^\wedge_\triangleleft)(\mathfrak{K}^\triangleleft + 2\alpha^2 \boldsymbol{I}_M)^{-1} \boldsymbol{S}^\top (\boldsymbol{U}_\triangleright - \mathfrak{B}_\triangleright) \\ &(\mathfrak{K}^\triangleright + 2\alpha^2 \boldsymbol{I}_M)^{-1} \mathfrak{k}_\triangleright(\boldsymbol{x}^\wedge_\triangleright) \end{aligned} \quad (27)$$

$$\Sigma_\diamond = \left[\Lambda k^\triangleleft - \mathfrak{k}_\triangleleft \left[\mathfrak{K}^\triangleleft + 2\alpha^2 \boldsymbol{I}_M\right]^{-1} \mathfrak{k}^\top_\triangleleft\right] \otimes \left[\Lambda k^\triangleright - \mathfrak{k}_\triangleright \left[\mathfrak{K}^\triangleright + 2\alpha^2 \boldsymbol{I}_M\right]^{-1} \mathfrak{k}^\top_\triangleright\right] \quad (28)$$

with objects $\mathfrak{B}_\triangleleft, \mathfrak{k}_\triangleleft, \mathfrak{K}^\triangleleft$, defined analogous to Equations (23) to (25). In fact, given our choice of kernel and a symmetric mean function, these objects are *identical* along the sub-space $\boldsymbol{x}_\triangleleft = \boldsymbol{x}_\triangleright$ in which all relevant evaluations take place. We will thus simplify to $\mathfrak{k}_\triangleleft = \mathfrak{k}_\triangleright = \mathfrak{k}$, etc. Note that the mean is not, in general, a symmetric matrix. This is the effect of our deliberate ignorance about the relationship between derivatives, which eliminates information about the commutativity of derivatives. It is easy, albeit somewhat ad hoc, to project the mean estimate into the space of symmetric matrices, using the projection operator $\Gamma = \frac{1}{2}(\boldsymbol{I} + \boldsymbol{T})$ with the transposition operator $\boldsymbol{T}\boldsymbol{x} = \boldsymbol{x}^\top$.

On a superficial level, the final result is a relatively simple extension of that of Hennig & Kiefel. But the derivations up to here considerably clarify the information content of prior and the two likelihood terms in the inference for the Hessian. In particular, it is now clear the inference scheme does not double count observations; in fact it *under*-counts, by considering a doubly large input space $\mathbb{R}^{N\times 1} \times \mathbb{R}^{1\times N} \simeq \mathbb{R}^{2N}$.

2.3. Estimating the inverse Hessian

We now have a joint Gaussian belief over the elements of the Hessian. Our belief over the *inverse* of this matrix, which we require for the Newton-Raphson direction, is not Gaussian. But a consistent point estimator for this direction can be constructed efficiently as $-B_\diamond^{-1}(\boldsymbol{x}^\curlyvee) \cdot \mu_\nabla(\boldsymbol{x}^\curlyvee)$ (where $\boldsymbol{x}^\curlywedge = \boldsymbol{x}^\curlywedge_\triangleleft = \boldsymbol{x}^\curlywedge_\triangleright$), by inverting the mean estimate $B_\diamond$, using the matrix inversion lemma. After symmetrisation, the mean estimate can be written as

$$B_\diamond = B_0 + \begin{pmatrix}\boldsymbol{V} & \boldsymbol{W}\end{pmatrix}\begin{pmatrix}0 & \boldsymbol{I}_M \\ \boldsymbol{I}_M & 0\end{pmatrix}\begin{pmatrix}\boldsymbol{V} \\ \boldsymbol{W}\end{pmatrix} \quad \text{with} \tag{29}$$

$$\boldsymbol{V} \equiv \mathfrak{k}(\mathfrak{K} + 2\alpha^2 \boldsymbol{I}_M)^{-1} \quad \text{and} \tag{30}$$

$$\boldsymbol{W} \equiv (\boldsymbol{U}_\triangleright - \mathfrak{B}_\triangleright) - \frac{1}{4}\mathfrak{k}(\mathfrak{K} + 2\alpha^2 \boldsymbol{I}_M)^{-1} \cdot [(\boldsymbol{U}_\triangleright - \mathfrak{B}_\triangleright)^\intercal + (\boldsymbol{U}_\triangleright - \mathfrak{B}_\triangleright)]. \tag{31}$$

Our estimate of the Newton-Raphson direction can be thus evaluated in $\mathcal{O}(N \cdot M^3)$. Of course, the number of previous evaluations M rises over time. But the algorithm moves along a trajectory, so old observations typically become irrelevant as the algorithm moves away from them. Analogously to the L-BFGS method (Nocedal, 1980), we simply impose a *memory limit* $M_{\max}$ and ignore all observations more than $M_{\max}$ steps away (see also Hennig & Kiefel, 2012).

2.4. Relation to classic quasi-Newton methods

For the noise-free limit of $\alpha \to 0$, there is a close conceptual connection to classic quasi-Newton methods, as shown in Hennig & Kiefel (2012): If we perform the updates in individual steps, re-setting uncertainty to the prior in every step, and set the length scales $\lambda \to \infty$, which amounts to a locally constant model for the Hessian (assuming the function is a quadratic), the signal variance to unit, $\theta = 1$, and the noise to zero $\alpha \to 0$, then Equation (21) reduces to Broyden's (1965) method, Equation (27) to Powell's (1970) symmetric Broyden method. If we further assume the function to be convex and chose each kernel $k_\triangleleft, k_\triangleright$ to be equal to B itself (a concept that does not easily generalise to non-constant Hessians), then we arrive at the DFP update formula (Davidon, 1959; Fletcher & Powell, 1963). From there, under the bijection $Y \leftrightarrow S, B \leftrightarrow B^{-1}$, we obtain the widely used BFGS formula (Broyden, 1969; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). All these methods share regularization terms which can be interpreted analogously to the derivations in this paper, as assuming independence between the objective's dependence on row and column-vectors. So, while the idea of treating separating vectors and co-vectors as separate objects as introduced in this paper may appear far-fetched at first sight, it is in fact an old concept at the heart of some of the most widely used optimization algorithms—it is just not obvious in their form. Certain classic algorithms are parametric, noise-free limit cases of the method described here.

2.5. Open issues and limitations

The previous sections provide a principled, linear cost solution to the issue of inferring the Newton-Raphson direction under noise. There are other problems of stochastic optimization not addressed here, most importantly the choice of step size. Even in the noise free case, a fixed step size of 1 is not optimal. Quasi-Newton methods use a *line search* for this purpose, but this paradigm is nontrivial to use under noise, where bisection algorithms are not applicable. It is also clear that with rising noise the algorithm learns ever more slowly and eventually has to become more conservative in its behaviour. The experiments in Section 3 ignore this old question and leave the step size at 1 throughout. A more fundamental question is in which regime it is actually useful to try to learn the Hessian: Clearly, in the limit of infinite noise a quasi-Newton method cannot learn anything and will simply follow the mean of the gradient. At the other end, for zero noise, it is well known that quasi-Newton methods substantially outperform simple gradient descent. The experiments shed light on this issue. The unsurprising bottom line: the right choice depends both on the noise level and the computational cost of the objective function. Very expensive functions justify the prize of comparably expensive methods like Hessian-free optimization, very cheap functions are optimized most efficiently by gradient descent. The method proposed in this paper covers the, perhaps most interesting, intermediate ground.

3. Experiments

In the following two experiments, a low-dimensional toy problem provides intuition and evaluates numerical precision in the convergence limit, a realistic high-dimensional problem investigates cost.

3.1. Numerical Precision: Minimizing a Gaussian bowl

Consider a two-dimensional optimization problem in form of a "Gaussian bowl": $f(\boldsymbol{x}) = -\exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{m})^\intercal P(\boldsymbol{x} - \boldsymbol{m}))$, with location $\boldsymbol{m} \in \mathbb{R}^2$ sampled from a unit bivariate Gaussian distribution and inverse scale $P \in \mathbb{R}^{2\times 2}$, sampled from a Wishart distribution with 3 degrees of freedom. Because this function has nontrivial gradient and Hessian (proportional to the first and second Hermitian polynomials, respectively), it is nontrivial for Newton-Raphson.

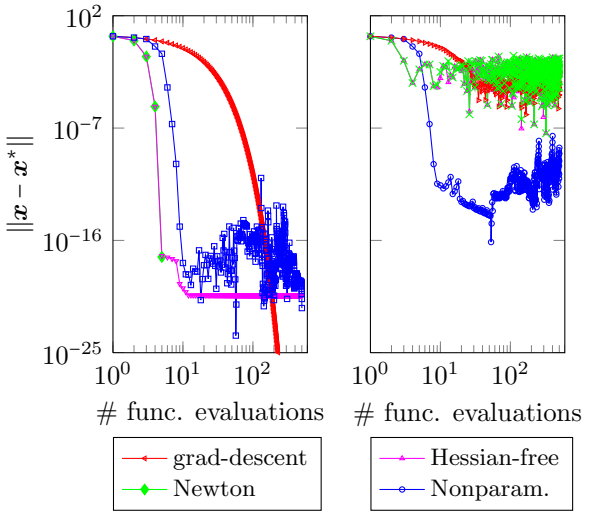

Figure 2. Performance on bowl problem, without noise (**left**) and with noise (**right**). Note shared double-logarithmic scale and large range of abscissa. The results for Newton and Hessian-free optimization largely overlap, the result for noise-free Newton jumps to zero after 5 evaluations.

Figure 2 shows the performance of a number of optimization algorithms on the "Gaussian bowl" problem: gradient descent, exact Newton-Raphson, Hessian-free optimization (Martens, 2010), which is essentially an elegant numerical implementation of exact Newton-Raphson, and the nonparametric probabilistic quasi-Newton algorithm. Gradient descent used the learning rate rule $\alpha_t = 500/t$, which was the best choice found in a rough grid search. Each algorithm performed one optimization without evaluation noise, and one with noise. Noise was independent Gaussian on each element of the gradient, with standard deviation 0.01, which corresponds to a relatively high signal to noise ratio of ~ 100. Noise vectors were sampled once per evaluation and shared by all algorithms. As in the mini-batch setting, the noise was kept constant throughout each call to Hessian-free optimization to allow a consistent inversion using conjugate gradients. Algorithms which calculate the Hessian directly (Newton-Raphson and Hessian-free optimization) were given access to the exact Hessian even where noise was added to the gradient. This creates an advantage for these methods, which they were not able to use: The log-log plot shows performance over the entire dynamic range of the optimization, from initialisation down to single precision. The noise-free quasi-Newton method, despite having no access to the Hessian, performs almost as well as the better informed Newton methods (the rough shape towards the end of optimization is due to numerical instabilities). More strikingly, it is the only method in this comparison that is robust to noise: All three other methods saturate roughly at the noise level, while the probabilistic method descends about 10 orders of magnitude below it.

It is not surprising that the other second-order methods can not deal well with noise, they were not designed for this purpose. And of course this experiment, with its low-dimensional setting, is not particularly close to most applications. But it offers insight into the qualitative difference between stochastic gradient descent's diffusive kind of convergence, and the kind of convergence afforded by actively modeling noise.

3.2. Numerical Cost: Training a deep belief network

A realistic large-scale setting is optimization of the weights of a neural network. Our implementation is based on that by Hinton and Salakhutdinov[1] (Hinton & Salakhutdinov, 2006), which trains a 3-layer feed-forward network on the MNIST dataset[2]. Pre-training by contrastive divergence was run without change as in the original code. To create a reasonably manageable optimization problem that could be run numerous times using various optimization algorithms, only the top (classification) layer weights were then optimized, which creates a $N = 20,010$ dimensional optimization problem. It is important to point out that this limitation was introduced only for experimental convenience – the algorithm described in this paper does in fact scale to much higher values of N (we have anecdotal experience with optimization problems in the range $N \sim 10^6$).

The MNIST training set consists of $60,000$ images. Figure 3.2, top, shows that the noise created by even splitting this dataset into two "mini"-batches is already considerable. Adjusting the size of the mini-batches thus allows varying both computation cost for, and evaluation noise on the objective function over wide ranges.

Figure 3.2, middle and right, shows optimization performance as a function of number of evaluations and computation time, respectively. The right plot should be taken with a grain of salt, since computation time changes with the complexity of the objective function, and with implementation of the optimizer. In this particular case the objective function is comparably cheap, making it harder for the nonparametric algorithm to use its conceptual advantages. To get a noise-free plot, the loss function achieved by the algorithms on smaller

[1] `http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html`

[2] `http://yann.lecun.com/exdb/mnist`

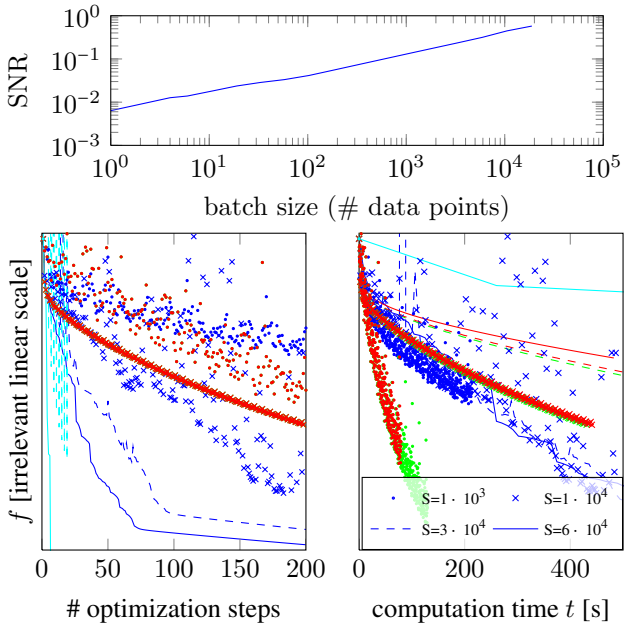

Figure 3. **Top:** signal to noise ratio for the DBN training problem, estimated by splitting the MNIST set into mini-batches of the indicated size and evaluating mean, standard deviation of f over the batches. **Bottom left and right:** f-values achieved by stochastic and regularised gradient descent (red and green, respectively, mostly overlapping), probabilistic quasi-Newton descent (blue), Hessian free-optimization (cyan) as a function of number of update steps and computation time, respectively, for varying batch sizes S. The Hessian-free method became unstable for all but the two largest S, a problem that might be solvable with more careful numerical engineering and should not be misunderstood as a criticism of this method.

(noisy) batches was re-evaluated on the whole, expensive dataset, i.e. the plotted function values were not observed in this form by the algorithms. The figure compares stochastic gradient descent to the nonparametric quasi-Newton method, as well as to a "regularized gradient descent", which simply uses the GP mean of Eq. (17) instead of the noisy gradient. The stark difference between this method and the quasi-Newton optimizer shows that the learned Hessian is nontrivial, despite the high dimensionality of the problem and the strong factorisation assumptions in the model. The plots also show results from Hessian-free optimization, implemented using MATLAB's efficient symmetric LQ solver (Barrett et al., 1994).

As anticipated (Section 2.5), these results reveal a nontrivial situation: The probabilistic method dominates stochastic gradient descent in terms of optimization performance per step, but its slightly higher cost is relevant when the objective function is very cheap. Obviously, in the limits of no / high evaluation cost for f, the cheapest / most elaborate optimizer always wins, respectively. But there is also a relevant region in between, where the additional efficiency of the probabilistic optimizer outweighs its slightly higher cost, while remaining cheaper than the (technically nonlinear) Hessian-free method. Of course, this region can be extended further by a more efficient implementation than our simple MATLAB version (note that the Hessian-free method uses C code).

4. Conclusion

Optimization methods for large-scale problems must be cheap, robust, and efficient. This paper presents a nonparametric robust extension of classic quasi-Newton methods which appears to be the first to fullfill all three of these requirements: It has linear cost, accounts for Gaussian noise, and approximates the quadratically efficient Newton Raphson method. Achieving this required a nontrivial nonparametric model, which explicitly ignores algebraic knowledge in exchange for lower cost. A simple MATLAB implementation will be released along with this paper, at `http://probabilistic-optimization.org`.

Acknowledgments

I am thankful to Martin Kiefel for ongoing helpful discussions on all aspects of probabilistic optimization. I would further like to thank an anonymous reviewer for helpful comments on a draft of this paper.

References

Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., J., Dongarra, Eijkhout, V., Pozo, R., Romine, C., and Van der Vorst, H. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994.

Bordes, A., Bottou, L., and Gallinari, P. SGD-QN: Careful quasi-Newton stochastic gradient descent. *J of Machine Learning Research*, 10:1737–1754, 2009.

Broyden, C.G. A class of methods for solving nonlinear simultaneous equations. *Math. Comp.*, 19(92):577–593, 1965.

Broyden, C.G. A new double-rank minimization algorithm. *Notices American Math. Soc*, 16:670, 1969.

Byrd, R.H., Chin, G.M., Neveitt, W., and Nocedal, J. On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM J. Optimization*, 21(3):977–995, 2011.

Davidon, W.C. Variable metric method for minimization. Technical report, Argonne National Laboratories, Ill., 1959.

Dennis, J.E. Jr and Morée, J.J. Quasi-Newton methods, motivation and theory. *SIAM Review*, pp. 46–89, 1977.

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

Fletcher, R. and Powell, M.J.D. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, 1963.

Genz, A. Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Statistics and Computing*, 14(3):251–260, 2004.

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

Hennig, P. and Kiefel, M. Quasi-Newton methods – a new direction. In *Int. Conf. on Machine Learning (ICML)*, volume 29, 2012.

Hinton, G.E. and Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Martens, J. Deep learning via Hessian-free optimization. In *International Conference on Machine Learning*, 2010.

Martens, J. and Sutskever, I. Learning recurrent neural networks with Hessian-free optimization. In *International Conference on Machine Learning*, 2011.

Nocedal, J. Updating quasi-Newton matrices with limited storage. *Math. Comp.*, 35(151):773–782, 1980.

Pearlmutter, B.A. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

Powell, M.J.D. A new algorithm for unconstrained optimization. In Mangasarian, O. L. and Ritter, K. (eds.), *Nonlinear Programming*. AP, 1970.

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

Robbins, H. and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Schraudolph, N.N. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

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