Generic Methods for Optimization-Based Modeling

Justin Domke
Rochester Institute of Technology

Abstract

"Energy" models for continuous domains can be applied to many problems, but often suffer from high computational expense in training, due to the need to repeatedly minimize the energy function to high accuracy. This paper considers a modified setting, where the model is trained in terms of results after optimization is truncated to a fixed number of iterations. We derive "backpropagating" versions of gradient descent, heavy-ball and LBFGS. These are simple to use, as they require as input only routines to compute the gradient of the energy with respect to the domain and parameters. Experimental results on denoising and image labeling problems show that learning with truncated optimization greatly reduces computational expense compared to "full" fitting.

1 Introduction

In supervised learning, one often wishes to fit a mapping from inputs $\mathbf{x} \in \mathbb{R}^N$ to outputs $\mathbf{y} \in \mathbb{R}^M$. This paper is inspired by the case where the mapping is defined by the minimization of some energy function, with

$$\mathbf{y}^*(\mathbf{x};\mathbf{w}) := \arg\min_{\mathbf{y}} E(\mathbf{y},\mathbf{x};\mathbf{w}). \quad (1)$$

Such mappings are common in structured prediction tasks such as denoising (Samuel and Tappen, 2009, Sun and Tappen, 2011, Barbu, 2009) or image labeling (Tappen et al., 2008, 2007). We are interested in setting the value of the weight vector $\mathbf{w}$ to minimize the empirical risk R on some dataset $\{(\mathbf{x}_1,\mathbf{y}_1),(\mathbf{x}_2,\mathbf{y}_2),...,(\mathbf{x}_m,\mathbf{y}_m)\}$. We assume that the optimization of the empirical risk will be done using some gradient-based optimization. So, the goal is to calculate

$$\frac{d}{d\mathbf{w}}R(\mathbf{w}) = \sum_{n=1}^{m} \frac{d}{d\mathbf{w}}L(\mathbf{x}_n,\mathbf{y}_n;\mathbf{w}) \quad (2)$$

$$= \sum_{n=1}^{m} \frac{d}{d\mathbf{w}}Q(\mathbf{y}^*(\mathbf{x}_n;\mathbf{w}),\mathbf{y}_n). \quad (3)$$

Here, L is the loss function, which is implicitly defined in terms of Q, which directly compares the predicted value $\mathbf{y}^*(\mathbf{x}_n;\mathbf{w})$ to the true value $\mathbf{y}_n$.

This paper considers computing $dL/d\mathbf{w}$ on a given training example. Implicit differentiation, the most traditional approach, has two steps: First, minimize the energy function to find $\mathbf{y}^*$. Then, recover $dL/d\mathbf{w}$ from the solution to a sparse linear system (Section 2). This approach can be problematic, as it is derived assuming that both the energy minimization and linear system are solved exactly, which is expensive in large problems. In practice, setting tolerances for these steps often involves a delicate trade-off between accuracy and computational expense (Sections 5-6).

This paper considers the case where the energy minimization is possibly *incomplete*. We let

$$\mathbf{y}^*(\mathbf{x};\mathbf{w}) = \underset{\mathbf{y}}{\text{opt-alg}}\, E(\mathbf{y},\mathbf{x};\mathbf{w}), \quad (4)$$

where opt-alg denotes the application of a fixed number of steps of some specific optimization algorithm. We wish to fit the empirical risk, even in cases where the energy optimization has not converged.

The contribution of this paper is to separate the problems of differentiating the optimization algorithm and the energy function, with no sacrifice in efficiency. Specifically, we develop methods to compute $dL/d\mathbf{w}$, when opt-alg is gradient descent, heavy-ball, or LBFGS. These methods require only routines to compute $dQ/d\mathbf{y}$, $dE/d\mathbf{w}$ and $dE/d\mathbf{y}$. The loss gradient $dL/d\mathbf{w}$ is computed in time proportional to that of computing $\mathbf{y}^*(\mathbf{x};\mathbf{w})$. Given these methods, it is easy to address a range of problems, with varying energy functions and optimization algorithms.

Related work includes Active Random Fields (Barbu,

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume XX of JMLR: W&CP XX.

2009), where a Fields of Experts denoising model (Roth and Black, 2009) is trained in the context of a fixed number of iterations of gradient descent, using a direct search technique. The drawback of this approach is high computational expense when there are many parameters. Sun and Tappen (2011) consider denoising with a model incorporating non-local interactions. They derive an algorithm to compute $dL/d\mathbf{w}$ for that particular energy, assuming that opt-alg is gradient descent. Using back-GD (derived below) with the same energy function yields a similar algorithm. Other work similar in spirit includes fitting graphical model parameters in terms of the marginals produced after a few message passing iterations (Domke, 2011, Stoyanov et al., 2011), fitting a sparse coding model in terms of a few iterations of coordinate descent (Gregor and LeCun, 2010), and approximating inference in restricted Boltzmann machines using a single iteration of mean-field (Larochelle and Murray, 2011).

In Sections 2-4, we concentrate on calculating $dL/d\mathbf{w}$ on some specific datum $(\mathbf{x}_n, \mathbf{y}_n)$, and so suppress it for simplicity. Thus, we use the shorthand notation of

$$L(\mathbf{w}) := L(\mathbf{x}_n, \mathbf{y}_n; \mathbf{w}) \quad (5)$$
$$E(\mathbf{y}; \mathbf{w}) := E(\mathbf{y}, \mathbf{x}_n; \mathbf{w}) \quad (6)$$
$$\mathbf{y}^*(\mathbf{w}) := \mathbf{y}^*(\mathbf{x}_n; \mathbf{w}) \quad (7)$$
$$Q(\mathbf{y}) := Q(\mathbf{y}, \mathbf{y}_n). \quad (8)$$

2 Implicit Differentiation

In this section, we review the most traditional way to fit continuous energy-based models. The basic result is the following theorem (Samuel and Tappen, 2009, Do et al., 2007).

Theorem 1. *Define* $\mathbf{y}^*(\mathbf{w}) = \arg\min_{\mathbf{y}} E(\mathbf{y}, \mathbf{w})$*, and* $L(\mathbf{w}) = Q(\mathbf{y}^*(\mathbf{w}))$*. Then, when all derivatives exist,*

$$\frac{dL}{d\mathbf{w}} = -\frac{\partial^2 E}{\partial \mathbf{w} \partial \mathbf{y}^T} \left(\frac{\partial^2 E}{\partial \mathbf{y} \partial \mathbf{y}^T} \right)^{-1} \frac{dQ}{d\mathbf{y}}, \quad (9)$$

where all quantities on the right hand side are evaluated at $\mathbf{y}^*(\mathbf{w})$ *and* $\mathbf{w}$*.*

Using this result, it is easy to calculate $dL/d\mathbf{w}$. First, perform the energy optimization to recover $\mathbf{y}^*$. Next, compute the loss gradient, and second-order derivative matrices of E, all at $\mathbf{y}^*$. Finally, solve the above linear system to recover he gradient. In practice, it is often complex or expensive to form the derivatives matrices, though this can be done in some cases (Samuel and Tappen, 2009, Tappen et al., 2007). Here, we focus on methods making use only of the gradients $\nabla_{\mathbf{y}} E$ and $\nabla_{\mathbf{w}} E$. If the linear system in Eq. 9 is solved using an iterative linear method such as conjugate-gradients, this is sufficient. This gives the implicit-CG algorithm,

Algorithm: (`implicit-cg`)

1. Perform the optimization to set
$$\mathbf{y}^* \leftarrow \arg\min_{\mathbf{y}} E(\mathbf{y}; \mathbf{w}).$$
2. Compute the loss $Q(\mathbf{y}^*)$ and the gradient $\mathbf{g} = \nabla_{\mathbf{y}} Q(\mathbf{y}^*)$.
3. Define the function
$$\text{hess-multiply}(\mathbf{v}) := \frac{\partial^2 E(\mathbf{y}^*; \mathbf{w})}{\partial \mathbf{y} \partial \mathbf{y}^T} \mathbf{v}.$$
4. Find $\mathbf{z} \leftarrow \text{conjugate-gradient}(\text{hess-multiply}, \mathbf{g})$.
5. Recover the parameter gradient
$$\frac{dL}{d\mathbf{w}} = -\frac{\partial^2 E(\mathbf{y}^*; \mathbf{w})}{\partial \mathbf{w} \partial \mathbf{y}^T} \mathbf{z}.$$

similar to that proposed by Do et al. in the context of hyper-parameter learning (Do et al., 2007).

By using efficient matrix-vector products, explicitly forming the second-order derivative matrices in steps 3 and 5 may be avoided; see the following section.

3 Finite Difference Matrix-Vector Products

Lemma 2. *Consider some differentiable function* $\mathbf{f} : \mathbb{R}^M \to \mathbb{R}^N$*. The product of the Jacobian of* $\mathbf{f}$ *with an arbitrary vector* $\mathbf{v}$ *is*

$$\frac{d\mathbf{f}}{d\mathbf{y}^T}\mathbf{v} = \lim_{r \to 0} \frac{1}{r} \Big(\mathbf{f}(\mathbf{y} + r\mathbf{v}) - \mathbf{f}(\mathbf{y}) \Big). \quad (10)$$

This essentially just states that the product of the Jacobian matrix $d\mathbf{f}/d\mathbf{z}^T$ with some arbitrary vector $\mathbf{v}$ is the derivative of the function $\mathbf{f}$ in the direction of $\mathbf{v}$. This result has long use in numerical analysis and machine learning (LeCun et al., 1993). It can be used to approximate matrix-vector products by taking some small but finite r. We follow Andrei (2009) in using $r = \sqrt{\epsilon}(1 + |\mathbf{y}|_\infty)/|\mathbf{v}|_\infty$, where ϵ is machine precision. Numerically superior results are given by using two sided differences, i.e. by using the approximation

$$\frac{d\mathbf{f}}{d\mathbf{y}^T}\mathbf{v} \approx \frac{1}{2r} \Big(\mathbf{f}(\mathbf{y} + r\mathbf{v}) - \mathbf{f}(\mathbf{y} - r\mathbf{v}) \Big), \quad (11)$$

which is accurate to $o(r^2)$. Higher-order approximations exist. For example, one can approximate the matrix-vector product to order $o(r^4)$ using four function evaluations (Boresi and Chong, 1991, Sec. 3.3).

In our implementations, we found that two sided differences as in Eq. 11 were accurate to several digits of precision, even when used recursively, as in the algorithms below. To reduce round-off error, one could instead use Pearlmutter's algorithm (1994), though this requires applying automatic differentiation techniques to the energy function code. Complex perturbations (Martins et al., 2003) could also be used to reduce round-off error, if the code for $\mathbf{f}$ supports complex numbers.

The two particular limits that we will make use of are

$$\frac{\partial^2 E}{\partial \mathbf{y} \partial \mathbf{y}^T}\mathbf{v} = \lim_{r \to 0} \frac{1}{r}\Big(\nabla_{\mathbf{y}} E(\mathbf{y} + r\mathbf{v}; \mathbf{w}) - \nabla_{\mathbf{y}} E(\mathbf{y}; \mathbf{w})\Big)$$
$$\frac{\partial^2 E}{\partial \mathbf{w} \partial \mathbf{y}^T}\mathbf{v} = \lim_{r \to 0} \frac{1}{r}\Big(\nabla_{\mathbf{w}} E(\mathbf{y} + r\mathbf{v}; \mathbf{w}) - \nabla_{\mathbf{w}} E(\mathbf{y}; \mathbf{w})\Big),$$

which follow from substituting $\mathbf{f}(\mathbf{y}) = \nabla_{\mathbf{y}} E(\mathbf{y}; \mathbf{w})$ and $\mathbf{f}(\mathbf{y}) = \nabla_{\mathbf{w}} E(\mathbf{y}; \mathbf{w})$ into Eq. 10.

Besides being efficient, this approach is very convenient; it is only necessary that the user provide routines to compute $\nabla_{\mathbf{y}} E$ and $\nabla_{\mathbf{w}} E$. For clarity, we will present all algorithms as taking matrix-vector multiplies, but use two-sided differences in implementation.

4 Back-Optimization

In deriving the implicit differentiation approach, it is assumed that both the minimization of the energy and the linear system are solved exactly. In practice, of course, one solves these to a finite accuracy. If the tolerances are too loose, the resulting loss gradient can be very inaccurate. In practice, tolerances must be set heuristically to try to avoid prohibitive expense without creating a uselessly inaccurate gradient.

The basic idea of back-optimization is to simply *define* the loss in terms of the results of an incomplete optimization. That is, define $\mathbf{y}^*(\mathbf{w}) = \text{opt-alg}_{\mathbf{y}} E(\mathbf{y}; \mathbf{w})$, where opt-alg denotes an operator that runs a given optimization algorithm for a specified number of iterations with a predetermined step-size.

One can imagine instead defining an operator that runs a given optimization algorithm until a specified convergence threshold is reached. However, such an approach has disadvantages. Most seriously, it can lead to $\mathbf{y}^*$ being non-differentiable with respect to $\mathbf{w}$.

Our first algorithm takes the derivative of a loss computed after a fixed number of iterations of gradient descent. Here, we adopt the notation of a left-pointing arrow to denote a (possibly intermediate) derivative of L with respect to some quantity. So, e.g., $\overleftarrow{\mathbf{w}}$ will eventually come to be equal to $dL/d\mathbf{w}$.

Algorithm: (`gradient-descent`)

1. Initialize $\mathbf{y}_0$.
2. For $k = 0, 1, ..., N-1$
   (a) $\mathbf{y}_{k+1} \leftarrow \mathbf{y}_k - \lambda \nabla_{\mathbf{y}} E(\mathbf{y}_k; \mathbf{w})$
3. $L \leftarrow Q(\mathbf{y}_N)$

Algorithm: (`back-gd`)

1. $\overleftarrow{\mathbf{y}^N} \leftarrow \nabla Q(\mathbf{y}_N)$
2. $\overleftarrow{\mathbf{w}} \leftarrow \mathbf{0}$
3. For $k = N-1, ...0$
   (a) $\overleftarrow{\mathbf{w}} \leftarrow \overleftarrow{\mathbf{w}} - \lambda \frac{\partial^2 E(\mathbf{y}_k; \mathbf{w})}{\partial \mathbf{w} \partial \mathbf{y}^T} \overleftarrow{\mathbf{y}_{k+1}}$
   $\overleftarrow{\mathbf{y}_k} \leftarrow \overleftarrow{\mathbf{y}_{k+1}} - \lambda \frac{\partial^2 E(\mathbf{y}_k; \mathbf{w})}{\partial \mathbf{y} \partial \mathbf{y}^T} \overleftarrow{\mathbf{y}_{k+1}}$
4. Return $\nabla_{\mathbf{w}} L = \overleftarrow{\mathbf{w}}$

Theorem 3. *After* `back-gd` *executes,* $\overleftarrow{\mathbf{w}} = dL/d\mathbf{w}$.

Proof. The basic idea is that, by the chain rule,

$$\frac{dL}{d\mathbf{w}} = \sum_{k=1}^{N} \frac{\partial \mathbf{y}_k^T}{\partial \mathbf{w}} \frac{dQ}{d\mathbf{y}_k}. \tag{12}$$

To calculate derivatives, we must calculate the two terms on the right-hand side on this equation. While $\frac{dQ}{d\mathbf{y}_N}$ is calculated directly, for $k \in \{0, ..., N-1\}$,

$$\frac{dQ}{d\mathbf{y}_k} = \frac{\partial \mathbf{y}_{k+1}^T}{\partial \mathbf{y}_k} \frac{dQ}{d\mathbf{y}_{k+1}} = \big(I - \lambda \frac{\partial^2 E(\mathbf{y}_k; \mathbf{w})}{\partial \mathbf{y} \partial \mathbf{y}^T}\big) \frac{dQ}{d\mathbf{y}_{k+1}}.$$

Similarly, we can calculate

$$\frac{\partial \mathbf{y}_{k+1}^T}{\partial \mathbf{w}} = -\lambda \frac{\partial^2 E(\mathbf{y}_k; \mathbf{w})}{\partial \mathbf{w} \partial \mathbf{y}^T}. \tag{13}$$

□

One might also fit parameters to initialize $\mathbf{y}_0$, in which case the derivative $\overleftarrow{\mathbf{y}_0}$ could be used.

Gradient descent has the advantage of simplicity. However, second-order optimization methods usually converge faster, sometimes dramatically so. Perhaps the simplest second-order method is *heavy-ball* (Polyak, 1964), or gradient descent with momentum. This is only slightly more complex than gradient descent, but often leads to faster optimization.

Theorem 4. *After* `back-hb` *executes,* $\overleftarrow{\mathbf{w}} = dL/d\mathbf{w}$.

Proof. As the immediate effects of $\mathbf{w}$ are only on $\mathbf{p}_k$,

$$\frac{dL}{d\mathbf{w}} = \sum_{k} \frac{\partial \mathbf{p}_k^T}{\partial \mathbf{w}} \frac{dQ}{d\mathbf{p}_k}. \tag{14}$$

Algorithm: (`heavy-ball`)

1. Initialize $\mathbf{y}_0$.
2. Initialize $\mathbf{p}_0 \leftarrow \mathbf{0}$
3. For $k = 0, 1, ..., N-1$
   (a) $\mathbf{p}_{k+1} \leftarrow -\nabla_{\mathbf{y}} E(\mathbf{y}_k; \mathbf{w}) + \gamma \mathbf{p}_k$
   (b) $\mathbf{y}_{k+1} \leftarrow \mathbf{y}_k + \lambda \mathbf{p}_{k+1}$
4. $L \leftarrow Q(\mathbf{y}_N)$

Algorithm: (`back-hb`)

1. $\overleftarrow{\mathbf{y}_N} \leftarrow \nabla Q(\mathbf{y}_N)$
2. $\overleftarrow{\mathbf{p}_N} \leftarrow \mathbf{0}$
3. $\overleftarrow{\mathbf{w}} \leftarrow \mathbf{0}$
4. For $k = N-1, N-2, ..., 0$
   (a) $\overleftarrow{\mathbf{y}_k} \leftarrow \overleftarrow{\mathbf{y}_{k+1}}$
   $\overleftarrow{\mathbf{p}_{k+1}} \leftarrow \overleftarrow{\mathbf{p}_{k+1}} + \lambda \overleftarrow{\mathbf{y}_{k+1}}$
   (b) $\overleftarrow{\mathbf{y}_k} \leftarrow \overleftarrow{\mathbf{y}_k} - \frac{\partial^2 E(\mathbf{y}_k;\mathbf{w})}{\partial \mathbf{y} \partial \mathbf{y}^T} \overleftarrow{\mathbf{p}_{k+1}}$
   $\overleftarrow{\mathbf{w}} \leftarrow \overleftarrow{\mathbf{w}} - \frac{\partial^2 E(\mathbf{y}_k;\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{y}^T} \overleftarrow{\mathbf{p}_{k+1}}$
   $\overleftarrow{\mathbf{p}_k} \leftarrow \gamma \overleftarrow{\mathbf{p}_{k+1}}$
5. Return $\nabla_{\mathbf{w}} L = \overleftarrow{\mathbf{w}}$

The first of the terms on the right hand side is

$$\frac{\partial \mathbf{p}_{k+1}^T}{\partial \mathbf{w}} = -\frac{\partial^2 E(\mathbf{y}_k; \mathbf{w})}{\partial \mathbf{w} \partial \mathbf{y}^T}, \tag{15}$$

while the second is

$$\frac{dQ}{d\mathbf{p}_k} = \frac{\partial \mathbf{p}_{k+1}^T}{\partial \mathbf{p}_k} \frac{dQ}{d\mathbf{p}_{k+1}} + \frac{\partial \mathbf{y}_k^T}{\partial \mathbf{p}_k} \frac{dQ}{d\mathbf{y}_k} \tag{16}$$

$$= \gamma \frac{dQ}{d\mathbf{p}_{k+1}} + \lambda \frac{dQ}{d\mathbf{y}_k}. \tag{17}$$

This requires the calculation of $dQ/d\mathbf{y}_k$, which is

$$\frac{dQ}{d\mathbf{y}_k} = \frac{\partial \mathbf{y}_{k+1}^T}{\partial \mathbf{y}_k} \frac{dQ}{d\mathbf{y}_{k+1}} + \frac{\partial \mathbf{p}_{k+1}^T}{\partial \mathbf{y}_k} \frac{dQ}{d\mathbf{p}_{k+1}} \tag{18}$$

$$= I \frac{dQ}{d\mathbf{y}_{k+1}} - \frac{\partial^2 E(\mathbf{y}_k; \mathbf{w})}{\partial \mathbf{y} \partial \mathbf{y}^T} \frac{dQ}{d\mathbf{p}_{k+1}}. \tag{19}$$

□

Finally, we consider LBFGS (Liu and Nocedal, 1989), based on Nocedal and Wright's notation (1999). In LBFGS, the previous M gradients are used to approximate the inverse Hessian at each inference iteration. Note, however that back-LBFGS stores all state vectors $\mathbf{y}_k$, and thus will have $O(ND)$ storage, rather than $O(MD)$, where D is the dimensionality of $\mathbf{y}$. In practice, this is a minor issue, as the motivation of these types of algorithms is to use a small value N. If necessary, state vectors and gradients could be recomputed as needed from logarithmic storage with only a constant factor increase in time complexity, using the technique known as checkpointing (Griewank, 1992).

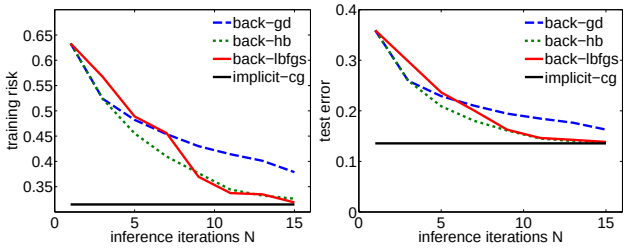


Figure 1: Binary Denoising results with varying numbers of inference iterations N. With few inference iterations, the back-optimization methods perform suboptimally, but with 15 iterations, heavy ball and LBFGS are similar to implicit.

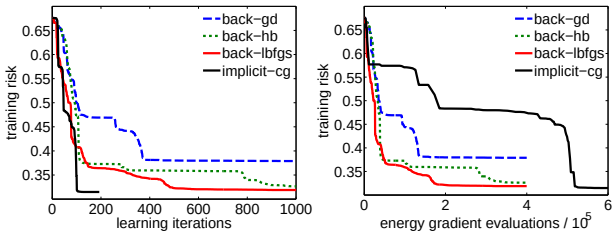


Figure 2: Binary denoising results, with $N = 15$ iterations used for all back-optimization methods. In terms of training risk vs. learning iterations, implicit-cg performs well. However, as it takes more energy gradient evaluations per learning iteration, it is much slower.

The derivation of back-LBFGS is quite tedious and so is omitted. It follows the same strategy as for gradient descent and heavy ball above, mechanically differentiating each step of the algorithm.

The rest of the paper reverts to the notation from the introduction, making explicit reference to both the inputs and outputs of the energy function, and loss function. In all of the following problems, learning is done by batch (non limited-memory) BFGS.

We use a step size $\lambda = 1$ for all algorithms. Note that for GD and HB, the effective step is fit in learning, by adjusting the magnitude of E. For HB, we set $\gamma = \frac{1}{2}$, while for LBFGS, we use $M = 5$.

Algorithm: (`lbfgs`)

1. Initialize $\mathbf{y}_0$.
2. For $k = 0, 1, ..., N-1$
   (a) $m \leftarrow \min(M, k)$
   (b) $\mathbf{g}_k \leftarrow \nabla_{\mathbf{y}} E(\mathbf{y}_k; \mathbf{w})$
   (c) $\mathbf{r}_k \leftarrow$ `get-dir`
   (d) $\mathbf{y}_{k+1} \leftarrow \mathbf{y}_k - \lambda \mathbf{r}_k$
3. $L \leftarrow Q(\mathbf{x}_N)$

Algorithm: (`get-dir`)

1. $\mathbf{q}_k \leftarrow \mathbf{g}_k$
2. For $i = k-1, ..., k-m$
   (a) $\mathbf{s}_i \leftarrow \mathbf{x}_{i+1} - \mathbf{x}_i$
   (b) $\mathbf{y}_i \leftarrow \mathbf{g}_{i+1} - \mathbf{g}_i$
   (c) $\rho_i \leftarrow 1 / \mathbf{y}_i^T \mathbf{s}_i$
   (d) $\alpha_i \leftarrow \rho_i \, \mathbf{s}_i^T \mathbf{q}_{i+1}$
   (e) $\mathbf{q}_i \leftarrow \mathbf{q}_{i+1} - \alpha_i \, \mathbf{y}_i$
3. If $k > 0$, $\gamma \leftarrow \frac{\mathbf{s}_{k-1}^T \mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^T \mathbf{y}_{k-1}}$,
   else $\gamma \leftarrow \gamma_0$.
4. $\mathbf{r}_{k-m} \leftarrow \gamma_k \mathbf{q}_{k-m}$
5. For $i = k-m, ..., k-1$
   (a) $\mathbf{r}_{i+1} \leftarrow \mathbf{r}_i + \mathbf{s}_i (\alpha_i - \rho_i \, \mathbf{y}_i^T \mathbf{r}_i)$

Algorithm: (`back-lbfgs`)

1. $\overleftarrow{\mathbf{y}_N} \leftarrow \nabla Q(\mathbf{y}_N)$
2. (Initialize $\overleftarrow{\mathbf{g}}$, $\overleftarrow{\mathbf{x}}$)
3. For $k = N-1, ..., 0$
   (a) $m \leftarrow \min(M, k)$
   (b) $\overleftarrow{\mathbf{r}_k} \leftarrow -\lambda \overleftarrow{\mathbf{y}_{k+1}}$
   $\overleftarrow{\mathbf{y}_k} \leftarrow \overleftarrow{\mathbf{y}_k} + \overleftarrow{\mathbf{y}_{k+1}}$
   (c) `back-get-dir`
   (d) $\overleftarrow{\mathbf{y}_k} \leftarrow \overleftarrow{\mathbf{y}_k} + \frac{\partial^2 E(\mathbf{y}_k; \mathbf{w})}{\partial \mathbf{y} \partial \mathbf{y}^T} \overleftarrow{\mathbf{g}_k}$
   $\overleftarrow{\mathbf{w}} \leftarrow \overleftarrow{\mathbf{w}} + \frac{\partial^2 E(\mathbf{y}_k; \mathbf{w})}{\partial \mathbf{w} \partial \mathbf{y}^T} \overleftarrow{\mathbf{g}_k}$

Algorithm: (`back-get-dir`)

1. For $i = k-1, ..., k-m$
   (a) $\overleftarrow{\mathbf{r}_i} \leftarrow \overleftarrow{\mathbf{r}_{i+1}} - \rho_i \mathbf{y}_i \mathbf{s}_i^T \overleftarrow{\mathbf{r}_{i+1}}$
   $\overleftarrow{\mathbf{s}_i} \leftarrow (\alpha_i - \rho_i \, \mathbf{y}_i^T \mathbf{r}_i) \overleftarrow{\mathbf{r}_{i+1}}$
   $\overleftarrow{\alpha_i} \leftarrow \mathbf{s}_i^T \overleftarrow{\mathbf{r}_{i+1}}$
   $\overleftarrow{\mathbf{y}_i} \leftarrow -\rho_i \mathbf{r}_i \mathbf{s}_i^T \overleftarrow{\mathbf{r}_{i+1}}$
   $\overleftarrow{\rho_i} \leftarrow -\mathbf{y}_i^T \mathbf{r}_i \mathbf{s}_i^T \overleftarrow{\mathbf{r}_{i+1}}$
2. $\overleftarrow{\mathbf{q}_{k-m}} \leftarrow \gamma \overleftarrow{\mathbf{r}_{k-m}}$
   $\overleftarrow{\gamma} \leftarrow \mathbf{q}_{k-m}^T \overleftarrow{\mathbf{r}_{k-m}}$
3. If $k > 0$
   (a) $\overleftarrow{\mathbf{s}_{k-1}} \leftarrow \frac{\mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^T \mathbf{y}_{k-1}} \overleftarrow{\gamma}$
   $\overleftarrow{\mathbf{y}_{k-1}} \leftarrow (\frac{\mathbf{s}_{k-1}}{\mathbf{y}_{k-1}^T \mathbf{y}_{k-1}} - \frac{2(\mathbf{s}_{k-1}^T \mathbf{y}_{k-1}) \mathbf{y}_{k-1}}{(\mathbf{y}_{k-1}^T \mathbf{y}_{k-1})^2}) \overleftarrow{\gamma}$
4. For $i = k-m, ..., k-1$
   (a) $\overleftarrow{\mathbf{q}_{i+1}} \leftarrow \overleftarrow{\mathbf{q}_i}$
   $\overleftarrow{\alpha_i} \leftarrow \overleftarrow{\alpha_i} - \mathbf{y}_i^T \overleftarrow{\mathbf{q}_i}$
   $\overleftarrow{\mathbf{y}_i} \leftarrow \overleftarrow{\mathbf{y}_i} - \alpha_i \overleftarrow{\mathbf{q}_i}$
   (b) $\overleftarrow{\rho_i} \leftarrow \overleftarrow{\rho_i} + \mathbf{s}_i^T \mathbf{q}_{i+1} \overleftarrow{\alpha_i}$
   $\overleftarrow{\mathbf{s}_i} \leftarrow \overleftarrow{\mathbf{s}_i} + \rho_i \mathbf{q}_{i+1} \overleftarrow{\alpha_i}$
   $\overleftarrow{\mathbf{q}_{i+1}} \leftarrow \overleftarrow{\mathbf{q}_{i+1}} + \rho_i \mathbf{s}_i \overleftarrow{\alpha_i}$
   (c) $\overleftarrow{\mathbf{y}_i} \leftarrow \overleftarrow{\mathbf{y}_i} - \frac{\mathbf{s}_i}{(\mathbf{y}_i^T \mathbf{s}_i)^2} \overleftarrow{\rho_i}$
   $\overleftarrow{\mathbf{s}_i} \leftarrow \overleftarrow{\mathbf{s}_i} - \frac{\mathbf{y}_i}{(\mathbf{y}_i^T \mathbf{s}_i)^2} \overleftarrow{\rho_i}$
   (d) $\overleftarrow{\mathbf{g}_i} \leftarrow \overleftarrow{\mathbf{g}_i} - \overleftarrow{\mathbf{y}_i}$
   $\overleftarrow{\mathbf{g}_{i+1}} \leftarrow \overleftarrow{\mathbf{g}_{i+1}} + \overleftarrow{\mathbf{y}_i}$
   (e) $\overleftarrow{\mathbf{y}_i} \leftarrow \overleftarrow{\mathbf{y}_i} - \overleftarrow{\mathbf{s}_i}$
   $\overleftarrow{\mathbf{y}_{i+1}} \leftarrow \overleftarrow{\mathbf{y}_{i+1}} + \overleftarrow{\mathbf{s}_i}$
5. $\overleftarrow{\mathbf{g}_k} \leftarrow \overleftarrow{\mathbf{g}_k} + \overleftarrow{\mathbf{q}_k}$

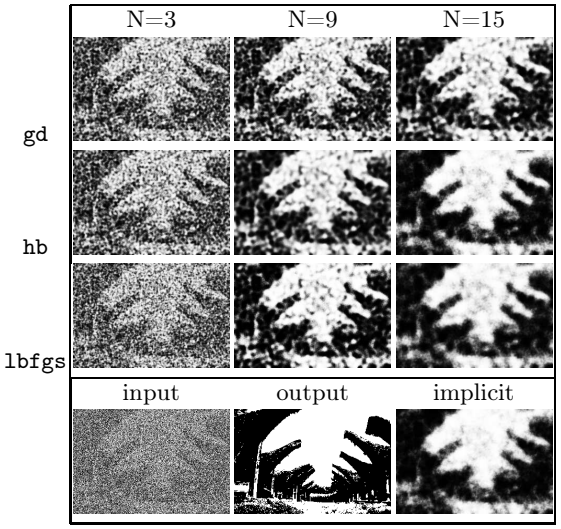


Figure 3: Example binary labeling results. Heavy ball, LBFGS, and implicit all achieve test errors in the range of 12.7%-13.0%. Though our purpose is not to compare different models, a CRF trained on a larger number of images from this same datatset achieves test errors of 12.6-14.0%, depending on the loss (Domke, 2010).

5 Example: Image Labeling

In this section, we compare the above techniques on a very simple model for structured prediction, similar to (a multiclass generalization of) the Logistic Random Field (Tappen et al., 2008). We claim no advantages to this model, other than being convenient benchmark for learning algorithms. The goal is to predict a label for each location i by means of a vector of scores $\mathbf{y}_i$. Let $\mathbf{y}^*$ be a set of indicator functions for the true labeling. (If the true label at location i is 5 then the 5th component of $\mathbf{y}_i^*$ is one and others are zero.) Then, we take the loss to be the sum of logistic losses over all locations,

$$Q(\mathbf{y}^*, \mathbf{y}) = -\sum_i \big(\mathbf{y}_i^T \mathbf{y}_i^* - \log \sum_j \exp y_{ij}^*\big). \qquad (20)$$

We use a quadratic model, taking a sum of the difference of linear transformations of score vectors at neighboring pairs, as well as the difference of linear transformations of a each score vector with the input features $\mathbf{x}_i$ at the same location, i.e.

$$E(\mathbf{y}, \mathbf{x}; W_1, W_2, V_1, V_2) = \sum_i ||V_1 \mathbf{y}_i - V_2 \mathbf{x}_i||_2^2 + \sum_{(i,j)} ||W_1 \mathbf{y}_i - W_2 \mathbf{y}_j||_2^2. \qquad (21)$$

The parameters of this problem are somewhat redundant, but this does no harm.

We will apply this to imaging problems, with one location for each pixel, and immediate (4-connected) pixels considered neighbors. Calculating the gradient of Q with respect to $\mathbf{y}^*$, or of E with respect to $\mathbf{y}$ or the four weight matrices is not hard, and so details are suppressed for space. As the energy is quadratic over $\mathbf{y}$, finding $\min_{\mathbf{y}} E$ could be done by standard least-squares methods. However, such methods are super-linear in the number of variables, and we prefer methods that readily generalize to more complex energies.

For each of the following problems, we first fit a standard (unstructured) logistic regression model to initialize V_2 and set $V_1 = I$. To provide a mild "smoothing" we initialize $W_1 = W_2 = \frac{1}{10} I$.

In a first experiment, we trained all methods on a small binary denoising dataset, designed so that it was practical to run implicit-CG to completion for comparison purposes. Following Domke (2010), we took 8 images from the Berkeley segmentation dataset, binarized them to obtain the true labeling. The noisy input values are then generated as $b(1-t^{1.25}) + (1-b)t^{1.25}$, where b is the true binary label, and $t \in [0,1]$ is random. Figures 1-3 show the results. For implicit-CG, a threshold of 10^{-4} for both optimization and conjugate gradients was selected (with the energy normalized by the number of pixels) after testing $10^{-1}, ..., 10^{-8}$ on a small problem and finding that this was the loosest threshold not leading to bad line directions and sub-optimal solutions.

Though in terms of the number of learning iterations, implicit-CG is the best performing method, this is no real advantage. In terms of energy function evaluations (time complexity), back-LBFGS is much faster. Further, despite taking more learning iterations, it finds a solution with essentially identical risk.

In a second experiment, we consider the semantic segmentation task. Here, the model is slightly generalized to include non-uniform weights. Specifically, each pair (i,j) has a vector of "edge features" $\mathbf{f}_{(i,j)}$, which weight the pairwise terms. Thus, we use the energy

$$E(\mathbf{y}, \mathbf{x}; W_1, W_2, V_1, V_2) = \sum_i ||V_1 \mathbf{y}_i - V_2 \mathbf{x}_i||_2^2 + \sum_{(i,j)} \exp(\mathbf{u}^T \mathbf{f}_{ij})\, ||W_1 \mathbf{y}_i - W_2 \mathbf{y}_j||_2^2. \qquad (22)$$

We use the Stanford backgrounds dataset(Gould et al., 2009a,b), at a resolution of approximately 80×107 (images vary slightly in resolution). We use a set of 42 features at each pixel consisting of a constant [1 feature], the pixel position [2 features], the LAB color space components [3 features], and a histogram of gradients (Dalal and Triggs, 2005) as computed by Dollár's toolbox (2011) [36 features]. As edge features $\mathbf{f}_{ij}$,

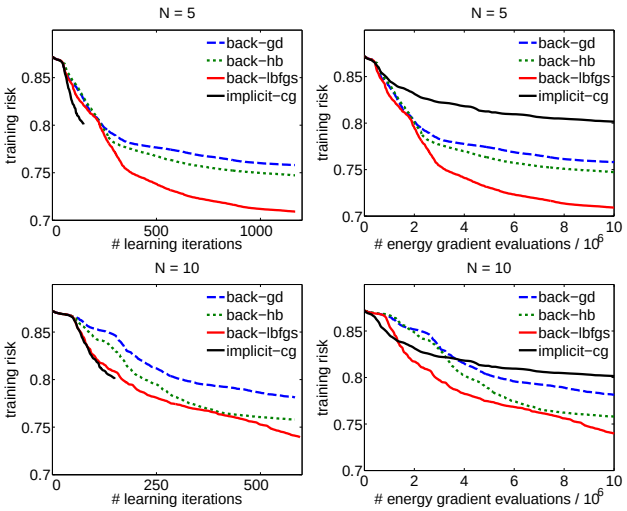


Figure 4: Scene labeling results, with each algorithm given 10^7 energy gradient evaluations, representing several days on computation time on an 8-core 2.26 GHz PC. Implicit-CG can complete few learning iterations within the allowed computation time. Back-LBFGS outperforms back-GD and back-HB.

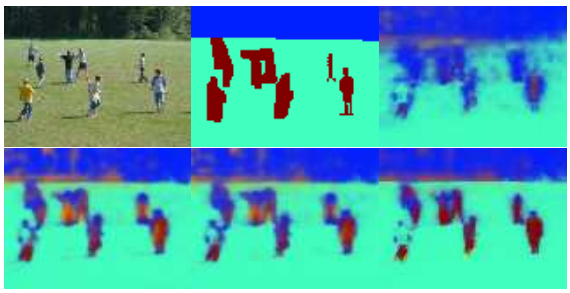


Figure 5: Scene labeling results. From top left: Input, truth, implicit-CG, back-GD, back-HB, and back-LBFGS, all with $N = 5$. (Best in color.)

we use both the l_2 norm of the RGB intensities at i and j and the maximum of a Sobel edge filter at i and j. Both of these are discretized into 10 bins, for a total of 20 edge features.

Results are shown in Figs. 4-5. Again, back-LBFGS is most successful in reducing the loss in a given amount of computation time.

6 Example: Fields of Experts Denoising

The next experiment is on learning a discriminative Fields of Experts (FoE) denoising model (Roth and Black, 2009). The energy function, corresponding to MAP inference given Gaussian noise, is

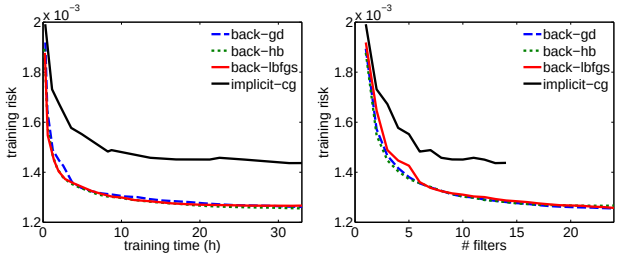


Figure 6: Training curves on denoising data. The back-optimization methods perform similarly, while implicit-cg is slow and suffers from imperfect inference convergence– sometimes even increasing in training risk after adding filters.

Table 2: Denoising performance (PSNR) on standard test images with a noise level of $\sigma = 25$. All are for 24 5×5 filter models with $N = 10$ inference iterations.

method	Barbara	Boats	House	Lena	Peppers	Average
`back-gd`	27.93	29.37	31.51	31.21	29.59	29.92
`back-hb`	27.81	29.33	31.50	31.14	29.51	29.86
`back-lbfgs`	28.05	29.37	31.52	31.25	29.59	29.96

$$\begin{aligned} E(\mathbf{y}, \mathbf{x}; \alpha, \beta, \mathbf{f}) &= e^{\alpha} \sum_k ||\rho(\mathbf{y} * \mathbf{f}_k)||_2^2 + e^{\beta} ||\mathbf{y} - \mathbf{x}||_2^2, \\ \rho(z) &= \log(1 + z^2), \end{aligned} \tag{23}$$

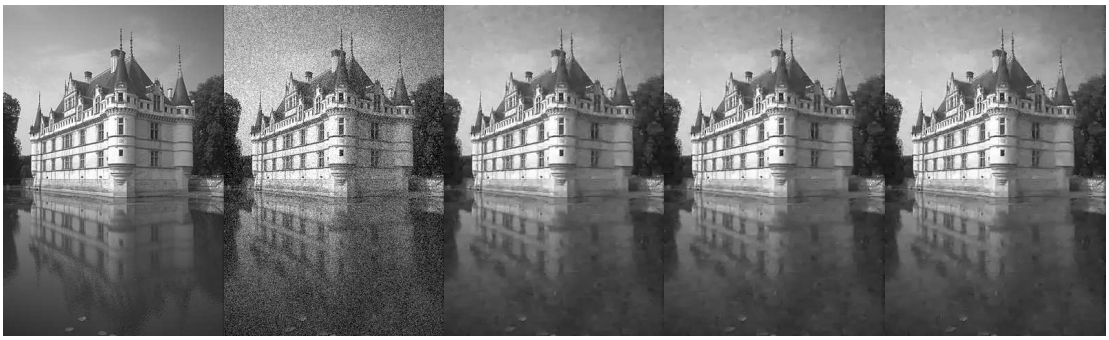
where $\{\mathbf{f}_k\}$ are a set of filters, and $*$ denotes convolution. We found little harm in using a single weight α for all filters rather than individual weights, as is more common. The derivatives $dE/d\alpha$ and $dE/d\beta$ are trivial, while $dE/d\mathbf{y}$ and $dE/d\mathbf{f}_k$ are well known (Zhu and Mumford, 1997, Eq. 14).

As a loss function, we use the squared difference of the predicted and true images

$$Q(\mathbf{y}^*, \mathbf{y}) = \sum_i (y_i^* - y_i)^2. \tag{24}$$

To reduce boundary effects, the sum in Eq. 24 is taken with a 5-pixel boundary ignored. At test time, the image boundaries are replicated by 50 pixels before denoising, then reduced. As in previous work (Barbu, 2009, Schmidt et al., 2010, Roth and Black, 2009, Sun and Tappen, 2011), we train on 40 and test on 68 images from the Berkeley dataset. We train on one 100×100 patch from each image.

Following Barbu (2009), we used a greedy training procedure, iteratively increasing the numbers of filters (all retrained at each step). Each new filter is initialized to a simple smoothing filter with $+8$ in the center, and -1 in the surrounding 8 pixels. We initialize $e^{\alpha} = 10^{-3}$



(a) Clean (b) Noisy (c) GD (d) HB (e) LBFGS

Figure 7: Results for denoising a test image. All algorithms use 24 5×5 filters and $N = 10$ iterations. Denoising a 321×481 image takes about 12 seconds on a 2.26 GHz PC. Differences between algorithms are small.

Table 1: Fields of Experts denoising results on 68 Berkeley segmentation images, with $\sigma = 25$ Gaussian noise, in terms of Peak Signal to Noise ratio (PSNR). Other papers (Sun and Tappen, 2011, Barbu, 2009) give results for standard denoising methods and more powerful model types. [1] - Linear systems solved using direct methods.

# filt.	size	training	inference	PSNR	citation
24	5x5	contrastive divergence	`gd-2500`	27.75	Roth and Black (2005)
8	3x3	contrastive divergence	gibbs sampling	27.95	Schmidt et al. (2010)
24	5x5	implicit-direct[1]	conjugate gradients (full)	27.86	Samuel and Tappen (2009)
13	5x5	random search	`gd-4`	28.21	Barbu (2009)
13	5x5	`back-gd`	`gd-10`	28.31	(ours)
24	5x5	`back-gd`	`gd-10`	28.39	(ours)
24	5x5	`back-hb`	`hb-10`	28.35	(ours)
24	5x5	`back-lbfgs`	`lbfgs-10`	28.39	(ours)

and $e^{\beta} = 1$, meaning the initial energy is dominated by the $||\mathbf{y} - \mathbf{x}||_2^2$ term.

Results are shown in Tables 1 and 2. Though our purpose is only to benchmark different learning algorithms, results are not competitive with state of the art denoising methods. Training time for each of our 13 (respectively 24) 5x5 filter models was approximately 13 (respectively 33) hours on a 8 core 2.26 GHz PC. Figure 6 shows training curves. Barbu (2009) finds a training time of around 12 days for his `gd-4` model, on a 8-core 2.4GHz PC (Extrapolating from a published time of 3 days for a `gd-1` model). Implicit-cg worked poorly on this problem. With a threshold of 10^{-5}, only 14 filters can be fit in the given amount of time and results appear to suffer from poor search directions.

The improvements over previous work are mostly because it is practical to train with more filters and inference iterations. LBFGS seemed to produce such small benefits because inference was initialized to the noisy image. If one initializes to zero, LBFGS and HB greatly outperform GD. Still, all methods perform better when initialized to the noisy image.

7 Discussion

With implicit-CG, too loose a threshold can lead to learning failure (Fig. 6). An advantage of back optimization methods is that one instead selects the number of inference iterations N, with a more straightforward tradeoff between speed and accuracy (Fig. 1).

The biggest advantage of back-LBFGS may simply be reliability. On some problems, implicit-CG may converge quickly enough or back-GD might perform nearly as well. Experimentally, however, back-LBFGS performs as well in all situations, and better in others.

The methods developed in this paper are likely to be applicable to the problem of hyper-parameter learning (Bengio, 2000, Do et al., 2007), where the goal would be to optimize performance hold-out set after a fixed number of iterations of training.

Future work might consider other inference methods. Another possibility would be to tune the parameters of the optimization at the same time as learning. For example, the performance of heavy-ball depends on the parameter γ. Thus, one might pursue $dL/d\gamma$, and include γ as a free parameter in the empirical risk.

References

Neculai Andrei. Accelerated conjugate gradient algorithm with finite difference Hessian/vector product approximation for unconstrained optimization. *J. Comput. Appl. Math.*, 230(2):570–582, 2009. ISSN 0377-0427.

Adrian Barbu. Training an active random field for real-time image denoising. *IEEE Transactions on Image Processing*, 18:2451–2462, 2009.

Yoshua Bengio. Continuous optimization of hyperparameters. In *IJCNN*, 2000.

Arthur Boresi and Ken Chong. *Approximate Solution Methods in Engineering Mechanics*. Elsevier Science Inc., 1991.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

Chuong Do, Chuan-Sheng Foo, and Andrew Ng. Efficient multiple hyperparameter learning for log-linear models. In *NIPS*, 2007.

Piotr Dollár. Piotr's image & video matlab toolbox, 2011. URL `http://vision.ucsd.edu/~pdollar/toolbox/doc/`.

Justin Domke. Implicit differentiation by perturbation. In *NIPS*, 2010.

Justin Domke. Parameter learning with truncated message-passing. In *CVPR*, 2011.

Stephen Gould, Richard Fulton, and Daphne Koller. Stanford background dataset, 2009a. URL `http://dags.stanford.edu/projects/scenedataset.html`.

Stephen Gould, Richard Fulton, and Daphne Koller. Decomposing a scene into geometric and semantically consistent regions. In *ICCV*, 2009b.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *ICML*, 2010.

Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1:35–54, 1992.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *AISTATS*, 2011.

Yann LeCun, Patrice Simard, and Barak Pearlmutter. Automatic learning rate maximization by on-line estimation of the Hessian's eigenvectors. In *NIPS*, 1993.

Dong Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

Joaquim Martins, Peter Sturdza, and Juan Alonso. The complex-step derivative approximation. *ACM Trans. Math. Softw.*, 29:245–262, 2003.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer, 1999.

Barak Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6:147–160, 1994.

Boris Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17, 1964.

Stefan Roth and Michael Black. Fields of experts: A framework for learning image priors. In *CVPR*, 2005.

Stefan Roth and Michael Black. Fields of experts. *International Journal of Computer Vision*, 82(2):205–229, 2009.

Kegan Samuel and Marshall Tappen. Learning optimized MAP estimates in continuously-valued MRF models. In *CVPR*, 2009.

Uwe Schmidt, Qi Gao, and Stefan Roth. A generative perspective on MRFs in low-level vision. In *CVPR*, 2010.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *AISTATS*, 2011.

Jian Sun and Marshall Tappen. Learning non-local range markov random field for image restoration. In *CVPR*, 2011.

Marshall Tappen, Ce Liu, Edward Adelson, and William Freeman. Learning gaussian conditional random fields for low-level vision. In *CVPR*, 2007.

Marshall Tappen, Kegan Samuel, Craig Dean, and David Lyle. The logistic random field – a convenient graphical model for learning parameters for MRF-based labeling. In *CVPR*, 2008.

Song-Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19:1236–1250, 1997.