Parameter Learning with Truncated Message-Passing

Justin Domke
Rochester Institute of Technology
justin.domke@rit.edu

Abstract

Training of conditional random fields often takes the form of a double-loop procedure with message-passing inference in the inner loop. This can be very expensive, as the need to solve the inner loop to high accuracy can require many message-passing iterations. This paper seeks to reduce the expense of such training, by redefining the training objective in terms of the approximate marginals obtained after message-passing is "truncated" to a fixed number of iterations. An algorithm is derived to efficiently compute the exact gradient of this objective. On a common pixel labeling benchmark, this procedure improves training speeds by an order of magnitude, and slightly improves inference accuracy if a very small number of message-passing iterations are used at test time.

1. Introduction

This paper focuses on learning graphical model parameters from data. One line of such research is based on marginal inference, usually accomplished by iterating "message-passing" updates until convergence occurs. Here, one fits the model to optimize some loss function, measuring how well predicted marginals agree with training data. This approach remains quite computationally challenging, as it requires re-solving the inference optimization in each learning iteration. For this reason, the state of the art for learned CRFs is generally to train a model at a significantly reduced image resolution[15, 13].

This paper identifies and resolves a major cause of this computational expense. The starting point is two observations (Section 3). First, during training, a tight convergence threshold on inference is necessary, as otherwise the computed loss gradient may not be a descent direction (Fig. 1). This tight threshold means that many message-passing iterations can be necessary, greatly slowing learning. However, at inference time, an extremely loose convergence threshold is sufficient to give good results (Table 2). Why spend so much effort in the learning stage exploring convergence levels that are irrelevant to the performance of the final model?

Our proposed solution is a modification of the learning objective. Existing loss functions are defined assuming inference has completely converged. If a loose convergence threshold is used, one obtains an inexact loss, and an inexact gradient. (A technical point should be emphasized here. The inexact loss and inexact gradient are inexact "in different ways". The inexact gradient is *not* the gradient of the inexact loss.)

This paper *defines* the loss function in terms of the approximate inference results obtained after message passing is "truncated" to a finite number of iterations. Computing the value of this loss function is trivial– simply perform the chosen number of message-passing iterations, and use the estimated marginals in place of the fully converged ones. One can efficiently compute the exact gradient, using an algorithm based on reverse-mode automatic differentiation.

With previous methods, truncating message-passing leads to an inaccurate gradient, and learning failure. The proposed solution gives an exact gradient for any number of message-passing iterations.

Experimental results show two benefits. First, in learning, an order of magnitude speedup occurs, as the number of message passing iterations can be reduced from hundreds to just a few ($e.g.\ 5-20$) with negligible loss of accuracy. Secondly, a small improvement in accuracy occurs in some circumstances. Namely, if one will only use a very small number ($e.g.\ 1-5$) of message-passing iterations at test time, the proposed method will be somewhat more accurate than a model fit for full convergence.

2. Background

2.1. Graphical Models and Inference

Consider a pairwise conditional random field (CRF) of the form

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_i \psi(y_i, \mathbf{x}) \prod_{(i,j)\in\mathcal{N}} \psi(y_i, y_j, \mathbf{x}),$$

where $\mathcal{N}$ is the set of all pairs in the graphical model, $\psi(y_i, \mathbf{x})$ and $\psi(y_i, y_j, \mathbf{x})$ are arbitrary positive functions,

and $Z(\mathbf{x})$ is a normalizing constant.

The goal of inference is to produce univariate and pairwise marginals, $p(y_i|\mathbf{x})$ and $p(y_i, y_j|\mathbf{x})$. Though computing these is intractable in high-treewidth models, they can be approximated by various methods. The focus here is on tree-reweighted (TRW) belief propagation[19]. In this algorithm one iterates the message updates

$$m_{t\to s}(y_s) \propto \sum_{y_t} \psi(y_t, \mathbf{x})\psi(y_t, y_s, \mathbf{x})^{1/\rho_{ts}} \cdot \frac{1}{m_{s\to t}(y_t)} \prod_{v\in N(t)} m_{v\to t}(y_t)^{\rho_{vt}},$$

over all pairs $(t, s) \in \mathcal{N}$. Here $N(t) = \{s : (t, s) \in \mathcal{N}\}$, and ρ_{ts} denotes the "appearance probability" of edge (t, s). In the case that all edge appearance probabilities are one, this reduces to loopy belief propagation. After messages have converged, approximate marginals μ can be obtained via

$$p(y_t|\hat{\mathbf{x}}) \approx \mu(y_t|\mathbf{x}) \propto \psi(y_t) \prod_{v\in N(t)} m_{v\to t}(y_t)^{\rho_{vt}}.$$

Bivariate marginals can be obtained by a similar, but somewhat more complex formula. One advantage of TRW is that if appropriate edge appearance probabilities are used, the algorithm is maximizing a concave function, and so convergence to a local optima is impossible. However, the messages may still fail to converge at all. Convergence in practice can be obtained by "damping" of the messages[18, p. 174]. On certain graphs, such as 4-connected grids, a careful ordering of the message-updates can guarantee convergence [10].

2.2. Loss Functions

Here, we are interested in fitting the parameters of a CRF to optimize some loss function, quantifying how accurate the model is on training data. If $\boldsymbol{\theta}$ are the parameters of the distribution, the objective function is

$$\sum_{(\hat{\mathbf{x}},\hat{\mathbf{y}})} L(\boldsymbol{\theta}, \hat{\mathbf{y}}, \hat{\mathbf{x}}).$$

Loss functions will be defined in terms of the approximate marginals μ. The experiments will focus on the univariate likelihood[8]

$$L(\boldsymbol{\theta}, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = -\sum_s \log \mu(\hat{y}_s|\hat{\mathbf{x}}).$$

Other marginal-based loss functions include the smoothed univariate classification accuracy[7] and the surrogate likelihood[17]. Note that not all loss functions are given in terms of approximate marginals. In particular, the pseudolikelihood can be computed directly from the factors ψ without inference, though it generally performs worse than inference-based loss functions in practice, sometimes dramatically so [16, 9, 14].

2.3. Gradient Calculation

A technical problem that emerges here is calculating the gradient of the loss with respect to the factors $\psi(y_s, \hat{\mathbf{x}})$ and $\psi(y_s, y_t, \hat{\mathbf{x}})$, namely

$$\frac{dL}{d\psi(y_s, \hat{\mathbf{x}})} \quad \text{and} \quad \frac{dL}{d\psi(y_s, y_t, \hat{\mathbf{x}})}, \tag{1}$$

taking into account the dependence of the predicted marginals μ on the factors ψ.

There are three major existing approaches to the calculating the derivatives $dL/d\psi(\cdot)$. The simplest is that, when using the surrogate likelihood, the loss gradient is available in closed form, given μ [17].

A second approach, applicable to a range of loss functions, is to apply implicit differentiation, leading to a matrix inversion problem[4]. Here, one performs inference, and then solves a large sparse linear system. A disadvantage of this approach is that solving this linear system can, in some cases, be more expensive than inference itself.

A third approach is Back-Belief Propagation (BBP)[5]. This is an algorithm based on applying reverse-mode automatic differentiation (RAD) to loopy belief propagation. BBP takes advantage of special properties of RAD when applied to fixed-point processes [11]. Namely, one need not store all intermediate values, as in standard automatic differentiation. The algorithm derived in this paper is essentially a tree-reweighted variant of BBP, generalized to the "truncated" setting where messages may not have fully converged.

A technical note: it can cause some confusion to pursue the derivatives in Eq. 1, since in practice the factors will usually be parameterized by some vector $\boldsymbol{\theta}$. Given the previous derivatives, it is easy to calculate the parameter gradient $dL/d\boldsymbol{\theta}$ via the chain rule[1]. This abstraction allows us to neglect the dependence of ψ on $\mathbf{x}$, which is application-specific, and often complex.

This paper explores the following issue: All three of the above methods assume that message-passing is iterated until convergence is exact. In practice, a reasonably tight convergence threshold is adequate. However, as we will see, the choice of this threshold can be quite problematic. Too loose, and inaccuracy in the gradient can cause

[1] Namely,

$$\frac{d}{d\boldsymbol{\theta}} L(\boldsymbol{\theta}, \hat{\mathbf{y}}, \hat{\mathbf{x}}) = \sum_s \sum_{y_s} \frac{dL(\boldsymbol{\theta}, \hat{\mathbf{y}}, \hat{\mathbf{x}})}{d\psi(y_s, \hat{\mathbf{x}})} \frac{d\psi(y_s, \hat{\mathbf{x}})}{d\boldsymbol{\theta}} + \sum_{(s,t)\in\mathcal{N}} \sum_{y_s, y_t} \frac{dL(\boldsymbol{\theta}, \hat{\mathbf{y}}, \hat{\mathbf{x}})}{d\psi(y_s, y_t, \hat{\mathbf{x}})} \frac{d\psi(y_s, y_t, \hat{\mathbf{x}})}{d\boldsymbol{\theta}}.$$

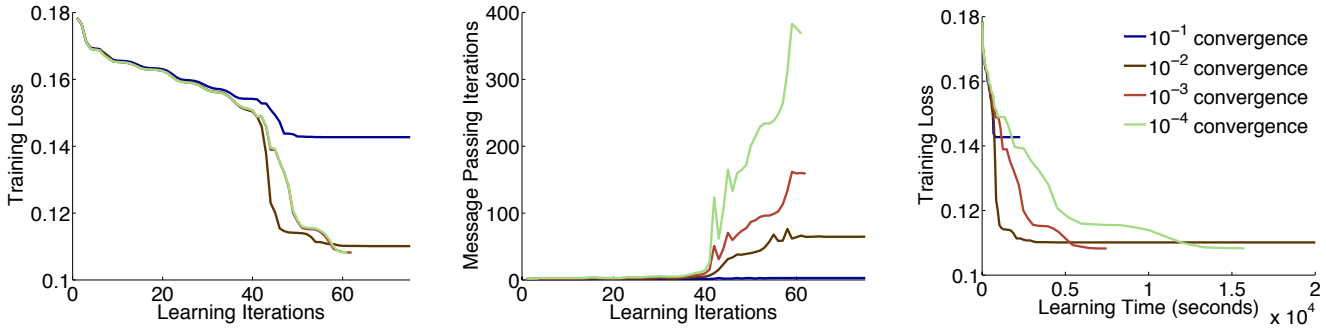

Figure 1. Learning with different message-passing convergence thresholds. These experiments use L-BFGS to optimize the univariate likelihood with tree-reweighted belief propagation on the MSRC-9 dataset. While a relatively tight threshold is necessary to avoid bad search directions, this results in hundreds of message passing iterations per learning step, greatly slowing the learning process.

Dataset	MSRC-9	
# Classes	9	
# Images Train	120	
# Images Test	120	
Resolution	213x320	
Model	$\prod_i \psi(y_i, \mathbf{x}) \prod_{(i,j)\in\mathcal{N}} \psi(y_i, y_j)$	
Linear Factors	$\psi(y_i = a, \mathbf{x}) = \mathbf{w}_a \cdot \mathbf{f}_i(\mathbf{x})$	
Features $\mathbf{f}_i$	Name	# Components
	Constant	1
	Position	2
	RGB	3
	HoG	36
	Total	**42**
Parameters	Type	# Parameters
	vertical $\psi(y_i, y_j)$	$81 = 9 \cdot 9$
	horizontal $\psi(y_i, y_j)$	$81 = 9 \cdot 9$
	$\mathbf{w}_a$	$378 = 42 \cdot 9$
	Total	**540**

Table 1. A summary of the experimental setting.

a bad search direction in the learning optimization. However, when made tight enough to avoid this problem, a huge number of message-passing iterations can result.

3. Motivating Example

Table 1 summarizes the experimental setting for all experiments in this paper, using the MSRC-9 dataset. While previous work on this dataset[12, 15] makes clear that sophisticated features are key to maximum performance, in the interests of experimental transparency, only a small-number of simple features are used here. These are a constant of 1, the vertical and horizontal position of each pixel, the RGB intensities of the pixel, and a histogram of oriented gradients[2], computed over an 8×8 patch of pixels, using the implementation of Dollar[3].

This section shows a example motivating why truncation might be helpful, comparing the performance of a tree-reweighted variant of the back-belief propagation algorithm (Section 4), with various convergence thresholds. Here, images are reduced to 25% resolution for speed. (The expense of parameter learning at full resolution with existing methods is, of course, the motivation for this paper.)

Figure 1 shows the results. We see that a convergence threshold of less than 10^{-3} is necessary, experimentally, to avoid getting stuck at a suboptimal solution. However, using a tight convergence threshold means a very large number of message passing iterations is necessary in later learning iterations, meaning that learning becomes very slow.

However, in Table 2, we see a surprise. Take the parameters learned with a tight convergence threshold, and simply evaluate the loss with different thresholds. Here, conversely, a loose convergence threshold performs extremely well.

Is it really necessary to spend hundreds of iterations in the learning stage exploring convergence levels that are irrelevant to the performance of the final model? The above results suggest that tight convergence is not necessary for good inference *results*, but simply for an accurate *gradient estimate*. A question naturally arises, then: consider the loss that is computed with a loose convergence threshold. Existing methods do not find the gradient of this. However, is it not possible to compute the exact gradient of this loss by other means?

Evaluation of Final Parameters			
Threshold	Mean Iterations	Loss	Error
10^{-1}	9.5	.101	.0336
10^{-2}	59.2	.106	.0333
10^{-3}	148.8	.107	.0335
10^{-4}	368.4	.108	.0335

Table 2. Evaluation of the final parameters learned with a 10^{-4} threshold. When the parameters are actually to be used, a loose convergence threshold is perfectly adequate.

4. Truncated Fitting

Consider performing a given number of message updates in a fixed order, followed by estimation of the marginals, and then computation of one of the previous loss functions. Each of these steps is differentiable, and so computing the gradient of the loss with respect to model parameters can be done, in principle, simply by application of reverse-mode automatic differentiation[6]. These techniques trace the execution of the original algorithm and then automatically reverse-propagate derivatives of the final loss over the execution trace, in time proportional to the original execution. Initial attempts to apply some standard autodiff tools (specifically ADOL-C[1] and RAD[6]) suffered from prohibitively poor performance, for two reasons. First, the reverse-propagation step was approximately 10-20 times slower than the manual method derived below, due to interpretation overhead in the reverse-propagation step. Second, autodiff tools lead to large memory requirements, due to storing *all* intermediate computational values. The method below only stores the computed messages $m_{s\to t}(y_t)$, meaning an order of magnitude lower memory usage.

For more efficiency, a manually derived method to reverse-propagate derivatives is shown as Algorithms 1 and 2. Following Eaton and Ghahramani[5], we use the notation of $\not{d}a$ to denote a (possibly intermediate) derivative of a with respect to the loss. Because the derivation is quite tedious, details are postponed to the Appendix[2]. The computational properties of Alg. 2 are now compared to message-passing inference, back-belief propagation (BBP) and a naive application of reverse-mode automatic differentiation (RAD). Suppose there are a total of M pairs of variables, each variable can take V possible values, and message passing takes N iterations. Message-passing inference will require at least on the order of MV^2N time, depending on the density of the graph. The same time is required for Alg. 2, BBP, and RAD. However, memory requirements differ. Alg. 2 requires MVN storage. This is less than RAD which, storing all intermediate values, will require MV^2N storage. However, this is more than BBP, which requires only MV.

In our implementation, the reverse-propagation step took approximately twice the amount of time of forward execution. The reverse-propagation algorithm makes use of the following lemma for "back-propagating with respect to normalization":

If $b_i = \frac{a_i}{\sum_j a_j}$, then

$$\frac{dL}{da_k} = \frac{dL}{db_k}\frac{1}{\sum_j a_j} - \sum_j \frac{dL}{db_j}\frac{a_j}{\left(\sum_j a_j\right)^2}. \qquad (2)$$

[2]Our Matlab/C++ implementation will be made freely available for the sake of reproducibility.

If steps involving the stack are omitted, and message-passing is iterated until convergence, Algorithm 2 reduces to a tree-reweighted variant of back-belief propagation[5]. Normal back-belief propagation follows from using $\rho_{ts} = 1$.

Algorithm 1 Truncated tree-reweighted belief propagation with message storing.

(Forward Propagation)
Repeat N times for all pairs (t, s):

- Push the messages $m_{t\to s}(y_s)$ onto a stack.
- $m^0_{t\to s}(y_s) \leftarrow \sum_{y_t} \psi(y_t)\psi(y_t, y_s)^{\rho_{ts}^{-1}} \frac{\prod_{v\in N(t)} m_{v\to t}(y_t)^{\rho_{vt}}}{m_{s\to t}(y_t)}$
- $m_{t\to s}(y_s) \leftarrow \frac{m^0_{t\to s}(y_s)}{\sum_{y'_s} m^0_{t\to s}(y'_s)}$

(Calculate Predicted Marginals)

- $\mu^0(y_s) \leftarrow \psi(y_s) \prod_{v\in N(s)} m_{v\to s}(y_s)^{\rho_{vs}}$
- $\mu(y_s) \leftarrow \frac{\mu^0(y_s)}{\sum_{y'_s} \mu^0(y'_s)}$
- $\mu^0(y_s, y_t) \leftarrow \psi(y_s)\psi(y_t)\psi(y_s, y_t)^{\rho_{st}^{-1}} \cdot \frac{\prod_{v\in N(s)} m_{v\to s}(y_s)^{\rho_{vs}}}{m_{t\to s}(y_s)} \frac{\prod_{v\in N(t)} m_{v\to t}(y_t)^{\rho_{vt}}}{m_{s\to t}(y_t)}$
- $\mu(y_s, y_t) \leftarrow \frac{\mu^0(y_s, y_t)}{\sum_{y'_s, y'_t} \mu^0(y'_s, y'_t)}$

5. Experiments

These experiments compare "full fitting" (i.e. learning based on running message passing to convergence and applying Alg. 2 without the stack as a tree-reweighted variant of BBP) to "truncated fitting" with various numbers of iterations. All algorithms used uniform edge-appearance probabilities of $\rho = .5$ on a 4-connected grid.

The motivation for this paper is the expense of existing methods for learning at high resolution. However, this very issue makes comparison to previous methods challenging. The set of experiments consider learning at low-resolution. This makes it possible to run all methods until convergence with multiple train/test splits, systematically comparing accuracy. A second set considers the motivating setting of learning at high resolution. Here, computational expense

Algorithm 2 Truncated back tree-reweighted belief propagation. (This is run after completion of message passing.)

(Initialize reverse-propagation structures.)

- Calculate loss L and partial derivatives $dL/d\mu$
- Calculate $dL/d\mu^0$ from $dL/d\mu$ via Eq. 2.
- $\not{d}\psi(y_s) \leftarrow \frac{dL}{d\mu^0(y_s)}\frac{\mu^0(y_s)}{\psi(y_s)} + \sum_{v\in N(s)}\sum_{y_v}\frac{dL}{d\mu^0(y_s,y_v)}\frac{\mu^0(y_s,y_v)}{\psi(y_s)}$
- $\not{d}\psi(y_s,y_t) \leftarrow \frac{1}{\rho_{st}}\frac{dL}{d\mu^0(y_s,y_t)}\frac{\mu^0(y_s,y_t)}{\psi(y_s,y_t)}$
- $\not{d}m^0_{v\to s}(y_s) \leftarrow \rho_{vs}\frac{dL}{d\mu^0(y_s)}\frac{\mu^0(y_s)}{m_{v\to s}(y_s)} + \sum_{t\in N(s)}\sum_{y_t}(\rho_{vs}-I_{v=t})\frac{dL}{d\mu^0(y_s,y_t)}\frac{\mu^0(y_s,y_t)}{m_{v\to s}(y_s)}$

(Back-propagate predicted marginals)
Repeat N times for all pairs (t,s), in reverse order:

- Calculate $\not{d}m^0_{t\to s}$ from $\not{d}m_{t\to s}$ via Eq. 2.
- $\not{d}m_{v\to t}(y_t) \stackrel{+}{\leftarrow} \sum_{y_s}(\rho_{vt}-I_{v=s})\not{d}m^0_{t\to s}(y_s)\frac{m^0_{t\to s}(y_s)}{m_{v\to t}(y_t)}$
- $\not{d}\psi(y_t) \stackrel{+}{\leftarrow} \sum_{y_s}\not{d}m^0_{t\to s}(x_s)\frac{m^0_{t\to s}(y_s)}{\psi(x_t)}$
- $\not{d}\psi(y_t,y_s) \stackrel{+}{\leftarrow} \frac{1}{\rho_{st}}\not{d}m^0_{t\to s}(y_s)\frac{m^0_{t\to s}(y_s)}{\psi(y_t,y_s)}$
- Overwrite $m_{t\to s}(x_s)$ by pulling from the stack.
- $\not{d}m_{t\to s}(x_s) \leftarrow 0$

(Output Results)

- Return L, $\frac{dL}{d\psi(y_s)} = \not{d}\psi(y_s)$, $\frac{dL}{d\psi(y_t,y_s)} = \not{d}\psi(y_t,y_s)$.

prevents running full fitting until a highly accurate solution is obtained with all methods. Instead, we compare how effectively different methods can reduce training loss with a given amount of training time.

Though the purpose here is to improve speed– not accuracy– the overall test errors of around 14.5% are in line with recent work on this dataset[15, 12].

5.1. Low-Resolution

Here, image resolution reduced by a factor of four. Feature maps are computed on the original image, and then reduced by bilinear interpolation. Training and test errors are measured by randomly splitting the 240 training images into half training and testing, averaging over five random splits. Parameters are learned to optimize the univariate likelihood using batch L-BFGS. Figure 2 summarizes the results, comparing the results of truncated fitting with a given number of iterations, to traditional full fitting, with the number of message-passing iterations fixed at evaluation time. We were surprised to see that performing full fitting and then truncating message-passing at test time performs quite well (in terms of accuracy). If one will only use 2-3 iterations at test time, there *is* a benefit to truncated fitting, but it is modest. At higher numbers of iterations, performance is nearly identical. However, due to the fact that full fitting must use a huge number of message-passing iterations to achieve the chosen convergence level of 10^{-4}, it is far slower. Figure 3 shows example test images and results.

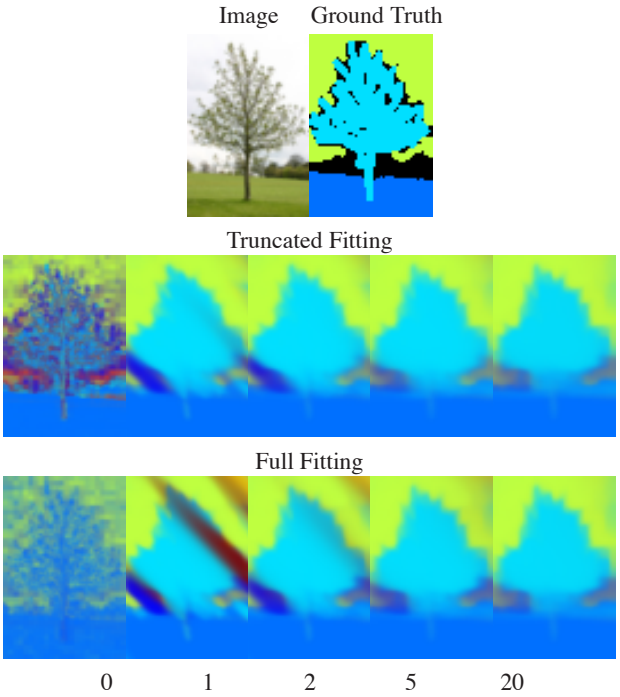

Figure 4. The (small) benefits of truncated training at test time. An example image on which truncated fitting for a given number of iterations gives different results from full fitting. (On many other images, there is little visible difference.)

5.2. Medium Resolution

A second experiment considers the images with resolution reduced by a factor of two. In Figure 5, the training loss is plotted as a function of the training time, with one day of training time given to each method on an 8-core 2.26 Ghz machine, with all cores used in parallel to compute the average loss over the 120 training images. After a full day of computation, full fitting can only obtain a loss reached by truncated fitting in around 4 hours. Truncated fitting with

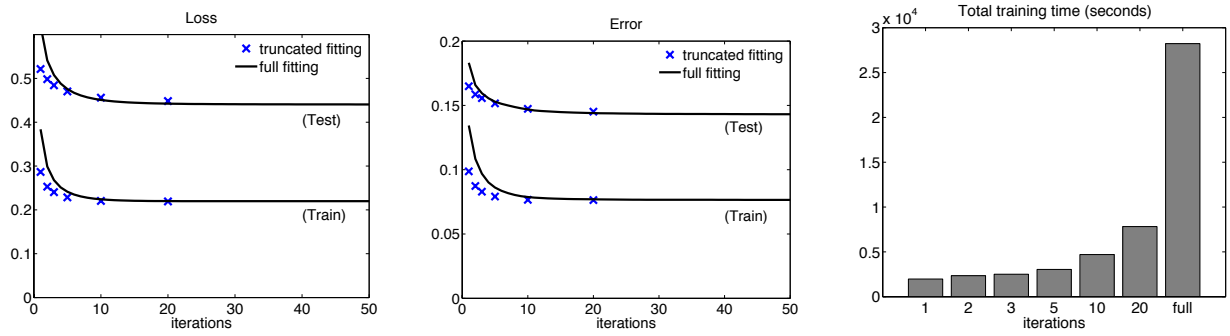

Figure 2. Quantitative evaluation of truncated fitting against full fitting at low resolution, evaluated with various numbers of iterations. For small numbers of test iterations, truncated fitting performs slightly better than full fitting. For larger numbers of iterations, the results are almost the same. However, truncated fitting is much faster for learning.

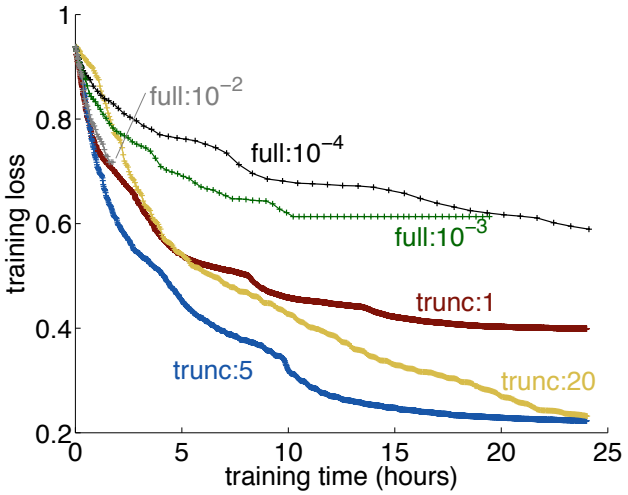

Figure 5. Training loss as a function of training time for high-resolution images, comparing full fitting with various convergence thresholds to truncated fitting with various numbers of message-passing iterations. One marker is plotted for each iteration. Full fitting with a threshold looser than 10^{-4} terminates early due to a bad search direction. Yet, full fitting with a threshold of 10^{-4} is slow to converge, due to the high computational cost per iteration.

one iteration rapidly reduces the loss, but becomes "stuck", presumably due to limits on the quality of inference available in one pass.

6. Discussion

This paper investigated the idea of fitting graphical models to optimize performance of a given message-passing algorithm after a fixed number of iterations. An algorithm was derived to compute the gradient, based on reverse-mode automatic differentiation, but with memory requirements from the order of MV^2N to MVN, for M pairs of variables with V possible values, and N passes of updates. With this algorithm one can, in practice, reduce the number of message-passing updates used for learning from hundreds to tens, yielding an order of magnitude speedup.

References

[1] Automatic differentiation by overloading in C++. http://projects.coin-or.org/ADOL-C.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[3] P. Dollar. Piotr's image & video toolbox for matlab. http://vision.ucsd.edu/ pdollar/toolbox/doc/, 2010.

[4] J. Domke. Learning convex inference of marginals. In *UAI*, 2008.

[5] F. Eaton and Z. Ghahramani. Choosing a variable to clamp. In *AISTATS*, 2009.

[6] D. Gay. Semiautomatic differentiation for efficient gradient computations. Technical Report, Sandia National Labs, 2004.

[7] S. Gross, O. Russakovsky, C. Do, and S. Batzoglou. Training conditional random fields for maximum labelwise accuracy. In *NIPS*, 2006.

[8] S. Kakade, Y. W. Teh, and S. Roweis. An alternate objective function for Markovian fields. In *ICML*, 2002.

[9] S. Kumar, J. August, and M. Hebert. Exploiting inference for approximate parameter learning in discriminative fields: An empirical study. In *EMMCVPR*, 2005.

[10] T. Meltzer, A. Globerson, and Y. Weiss. Convergent message passing algorithms - a unifying view. In *UAI*, 2009.

[11] B. A. Pearlmutter and H. E. Link. Algorithmic differentiation through convergent loops. Technical Report, 2002.

[12] F. Schroff, A. Criminisi, and A. Zisserman. Object class segmentation using random forests. In *BMVC*, 2008.

[13] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, 2008.

[14] C. Sutton and A. Mccallum. Piecewise training of undirected models. In *UAI*, 2005.

[15] J. Verbeek and B. Triggs. Scene segmentation with conditional random fields learned from partially labeled images. In *NIPS*, 2008.

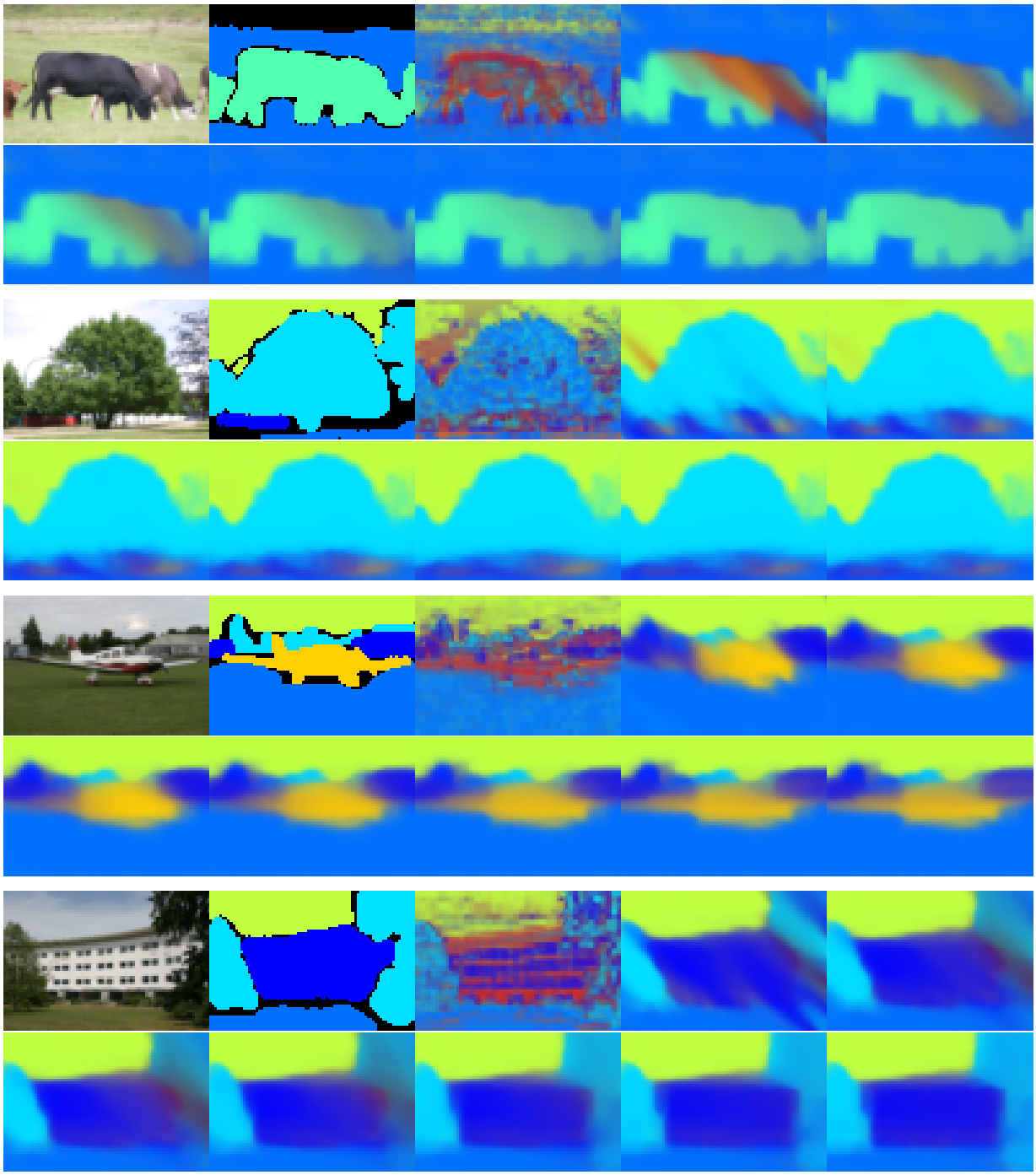

Figure 3. Low resolution test images showing, in order, the ground-truth labeling (unlabelled regions black), truncated fitting with 0, 1, 2, 3, 5, 10, and 20 iterations, and full fitting. Truncated fitting with 10-20 iterations is very similar to full fitting. (**Best in color.**)

[16] S. V. N. Vishwanathan, N. Schraudolph, M. Schmidt, and K. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML*, 2006.

[17] M. Wainwright. Estimating the "wrong" graphical model: Benefits in the computation-limited setting. *Journal of Machine Learning Research*, 7:1829–1859, 2006.

[18] M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2):1–305, 2008.

[19] M. J. Wainwright, T. Jaakkola, and A. S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.