Learning Cost-Aware, Loss-Aware Approximate Inference Policies for Probabilistic Graphical Models

Veselin Stoyanov
Johns Hopkins University
Baltimore, MD
ves@cs.jhu.edu

Jason Eisner
Johns Hopkins University
Baltimore, MD
jason@cs.jhu.edu

Abstract

Probabilistic graphical models are typically trained to maximize the likelihood of the training data and evaluated on some measure of accuracy on the test data. However, we are also interested in learning to produce predictions quickly. For example, one can speed up loopy belief propagation by choosing sparser models and by stopping at some point before convergence. We manage the speed-accuracy tradeoff by explicitly optimizing for a linear combination of speed and accuracy. Although this objective is not differentiable, we can compute the gradient of a smoothed version.

1 Introduction

Probabilistic graphical models (PGMs), such as Bayesian networks (BNs) and Markov random fields (MRFs), are commonly used for structured prediction. However, inference in PGMs is often slow. One can speed it up by approximation, but this can degrade accuracy.

We are interested in learning model structures and parameters that support fast approximate inference that remains (fairly) accurate. This is a test-time speed-up that requires an investment at training time.

In previous work [6], we were concerned with training for approximate inference. We argued that when PGMs may have incorrect model structure or will be used with approximate inference or decoding at test time, then their parameters should be robustly estimated by optimizing the training loss that is achieved *in the presence of all approximations*. In other words, one should perform empirical risk minimization (ERM) under the same approximations as those under which the system will be tested. We chose loopy belief propagation (BP) as our approximate inference strategy. To tune the graphical model parameters, we used back-propagation to compute the gradient of the output loss. We found that this mode of training indeed led to more accurate models than traditional maximization of the approximated log-likelihood. It also permitted speedups by running fewer iterations of BP. The performance of our ERM-trained models remained robust to this further approximation (and to misspecified model structure), whereas log-likelihood trained models degraded more rapidly.

In this paper we extend this work by *including speed as an explicit term in our training objective*, so that we learn parameters that balance speed and accuracy. In our BP setting, training choices can affect speed in two ways. First, we can accelerate inference by structure learning: e.g., in a *sparse* graphical model, fewer messages have to be passed. Second, we now expand our training-time optimization problem and introduce additional parameters that select an inference *strategy*; these *policy parameters* are tuned jointly with the model parameters. Our present experiments consider only a modest policy space, focusing on learning a dynamic stopping criterion for belief propagation.

In previous work we showed that in the computationally-sensistive setting, it is important to match the training and test conditions, including training to minimize the loss rather than to maximize the likelihood. In this work, we extend our experimental evaluation with more learning settings and by introducing mismatch in the order of the model. Our experimental results confirm previous observations. We then learn a sparse model and a dynamic stopping criterion. Unfortunately, it turned out that our current synthetic dataset did not need a dynamic stopping criterion, since it was possible to learn to perform accurate predictions even under a hard limit of two iterations. However, experiments show that including our sparsity-inducing technique reduces the number of edges by an order of magnitude while affecting the corresponding accuracy only modestly.

2 Preliminaries

Training Objective. We assume a supervised or semi-supervised learning setting. Each training example (x^i, y^i) consists of an input x^i and a desired output y^i. For purposes of this paper, our system's goal is to predict y^i from x^i. $\ell(y, y^i)$ defines the task-specific loss that the system would incur by outputting y. For the computationally-limited setting, we consider a *speed-augmented loss function* such as $\ell(y, y^i) + \lambda t_{\text{run}}(x^i)$, where $t_{\text{run}}(x^i)$ is the time that the model required to come up with the prediction. Here $\lambda > 0$ defines the tradeoff between speed and accuracy, making this a multi-objective optimization. By varying λ, we can sweep out the Pareto frontier.

Markov Random Field. An undirected graphical model, or **Markov random field** (MRF), is defined as a triple $(\mathcal{X}, \mathcal{F}, \Psi)$. $\mathcal{X} = (X_1, \ldots, X_n)$ is a sequence of n random variables; we use $\mathbf{x} = (x_1, \ldots, x_n)$ to denote a possible assignment of values. $\mathcal{F}$ is a collection of subsets of $\{1, 2, \ldots, n\}$; for each $\alpha \in \mathcal{F}$, we will write $\mathbf{x}_\alpha$ to denote an assignment to the subset of variables $\mathcal{X}_\alpha = \{X_j : j \in \alpha\}$.

Finally, $\Psi = \{\psi_\alpha : \alpha \in \mathcal{F}\}$ is a set of **factors**, where each factor ψ_α is some function that maps each $\mathbf{x}_\alpha$ to a **potential** value in $[0, \infty)$. This function depends implicitly on the global parameter vector θ. The probability of a given assignment $\mathbf{x}$ is given by

$$p_\theta(\mathcal{X} = \mathbf{x}) = \frac{1}{Z} \prod_{\alpha \in \mathcal{F}} \psi_\alpha(\mathbf{x}_\alpha) \tag{1}$$

where Z is chosen to normalize the distribution.

Each of our training inputs x^i is a set of "input" random variables with observed values, and the corresponding training output y^i is a set of "output" random variables, also observed. There may be additional hidden variables.

Loopy Belief Propagation. Inference in general MRFs is intractable. We focus in this paper on a popular approximate inference technique, **loopy belief propagation** (BP) [4]. BP uses iterative updates of messages and terminates when it either converges (i.e., the maximum message change, or residual, falls under a threshold) or a maximum number of iterations is reached. Sets of variable and factor marginals called beliefs are computed from the final messages. Beliefs can be used to compute gradients of the log-likelihood for training and used to produce predictions during testing.

Log-Likelihood Maximization. MRFs are typically trained instead by maximizing the log-likelihood of a given training set $\{(x^i, y^i)\}$. The log-likelihood $\log p_\theta(x^i, y^i)$, or the conditional log-likelihood $\log p_\theta(y^i \mid x^i)$, can be found by considering two slightly different MRFs, one with (x^i, y^i) both observed and one with only the conditioning events (if any) observed. An approximation of the $\log Z$ of each MRF can be computed by running BP and combining the beliefs into a quantity called the **Bethe free energy** that approximates $-\log Z$ [9] The gradient of the Bethe free energy is closely connected to the beliefs. Training MRFs based on the Bethe free energy approximation has been shown to work relatively well in practice [8, 7].

Empirical Risk Minimization. Maximizing log-likelihood is appropriate when it gives a good estimate of the true distribution (e.g., the model structure and training data are adequate), and furthermore that distribution will be used for exact inference and decoding (i.e., the system will actually make the minimum-risk decision given its input data). Under other conditions, we have argued that one should *train* to minimize risk for the actual inference and decoding procedures that will be used at test time [6]. Suppose we have f_θ, a family of decision functions parameterized by θ. In our case, θ gives the feature weights, and f_θ computes the result of y^* of approximate BP inference followed by some decoding procedure. We then use the loss function $\ell(y^*, y^i)$ to evaluate whether the decoder's output y^* was good. We should select θ to minimize the expected loss under the true data distribution over (x, y). In practice, we do not know the true data distribution, but we can do **empirical risk minimization**, taking the expectation over our sample of (x^i, y^i) pairs.

To determine the gradient of $\ell(f_\theta(x^i), y^i)$ with respect to θ, we employ automatic differentiation in the reverse mode [2]. The intuition is that our entire test-time system, regardless of how its constructed or the approximations that it uses, is nothing but a sequence of elementary differentiable operations. If intermediate results are recorded during the computation of the function (the forward pass), we can then compute the partial derivative of the loss with respect to each intermediate quantity, in the reverse order (the backward pass). At the end, we have accumulated the partials of the loss with respect to each parameter θ. A detailed explantation of our back-propagation algorithm as well as complete equations can be found in [6] .

Gradient information from the above procedure can be used in a local optimization method to minimize training test loss. In this paper we use stochastic meta descent (SMD) [5].

3 Learning Computationally Aware Models

We are interested in learning models that run fast during testing, even at some cost to accuracy. In the context of BP for PGMs, we can achieve speed-up by stopping approximate inference early (i.e., computing a cruder approximation) and by learning sparser models (i.e., models with fewer edges).[1]

Learning Through ERM. In previous work, we argued that one should learn through minimizing training loss under the same approximations as those that will be used during training. We hypothesise this is a particularly effective way to train when the quality of the approximation during testing degrades due to computational limitations.

Learning a Dynamic Stopping Policy. In BP, early stopping can be achieved by either imposing a limit on the number of BP iterations or terminating when the change of maximum message residual is below a certain threshold value. We propose a way for the model to decide to stop dynamically as a function of the computation state after each BP iteration. At test time, the system relies on a parameterized function of the computation state to decide when to stop the computation. During training we learn the parameters of the termination function **together** with the parameters of the model. More precisely, after each iteration, we compute a feature vector ϕ of the computation state including features such as the maximum residual value and how many BP iterations have been run. During testing, we terminate as soon as $w \cdot \phi > 0$, where w is a parameter vector. During training, we seek MRF parameters θ and policy parameters w to minimize speed-augmented loss.

To make the training objective differentiable, we replace the threshold 0 with the random threshold $\Theta \sim \mathrm{Logistic}(0, s)$, where $\mathrm{Logistic}(0, s)$ is the logistic distribution with cumulative density function given by $F(x; s) = 1/(1 + e^{-x/s})$. Our output y depends on the number of BP iterations. Let y_t be the output if we stop after t iterations. The stopping probability then is $p(\mathrm{st}_t) = p(w \cdot \phi_t > \Theta) = F(w \cdot \phi_t; s)$. The expected speed-augmented loss of our output y on the training example (x^i, y^i) is then $\mathbb{E}_T[\ell(y, y^i) + t_{\mathrm{run}}(x^i)] = \sum_{t=1}^{t_{\max}} \left(\prod_{t'=1}^{t-1}(1 - p(\mathrm{st}_{t'}))\right) p(\mathrm{st}_t)(\ell(y_t, y^i \lambda t)$. The parameter s controls the variance of the logistic distribution and is gradually decreased during training.

Learning Sparse Models to Speed up Computation. We wish to reward our learner for dropping factors from the MRF (rather like group lasso). The runtime *per iteration* of BP is proportional to the number of factors in the graph. We therefore wish to prune "weak" factors that will have little influence on the result. We measure the *strength* of the factor ψ_α as $S_\alpha = \sum_{\mathbf{x}_\alpha} (\log \psi_\alpha(\mathbf{x}_\alpha))^2$, where the sum is over all assignments $\mathbf{x}_\alpha$ to $\mathcal{X}_\alpha$. At training time, we prune "softly" by considering α to be fractionally present to the degree $\tau(S_\alpha) \in (0, 1)$, where τ is the sigmoid-like function $\tau(x) = \frac{2}{1+\exp(1/x)^s} \in (0, 1)$.[2] This fractional presence means that the runtime of an iteration is measured as $\sum_{\alpha \in \mathcal{F}} \tau(S_\alpha)$, and that we redefine equation (1) as

$$p_\theta(\mathcal{X} = \mathbf{x}) = \frac{1}{Z} \prod_{\alpha \in \mathcal{F}} \psi_\alpha(\mathbf{x}_\alpha)^{\tau(S_\alpha)} \tag{2}$$

This keeps our training objective differentiable, but gives it an incentive to learn weaker factors because they are considered to be faster. At test time, we do hard pruning of weak factors, dropping α from the MRF iff $\tau(S_\alpha) < 1/2$.

4 Experiments

We perform experiments on synthetic data. We generate a random MRF with 50 binary random variables (RVs) and 200 factors each of which includes interactions between four random variables.

[1]An extension that we leave for future work is to achieve speed up by learning prioritization and pruning heuristics for message updates. Fixed prioritization heuristics have been shown to be useful to speed up BP [1].

[2]One should gradually increase s during training, but for present experiments we simply fix $s = 1$.

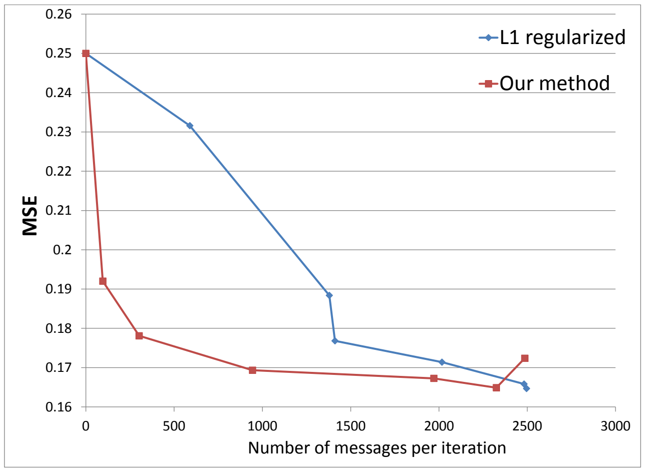

Figure 1: Accuracy for different levels of speed (number of messages sent per BP iteration per test example). Speed-up is achieved by increasing the strength of the L1-regularizer for the baseline method and by increasing the value of λ for our method.

We randomly sample factor weights from a normal distribution and then exponentiate them. We randomly designate 1/3 of the RVs as input, 1/3 as hidden and 1/3 as output. We use Gibbs sampling to generate 1000 training and 1000 test data points. For evaluation we use mean squared error (MSE) on the output marginals.

During training we pretend that the true model structure is unknown, so training starts with a complete graph of binary factors (only) as a reasonable and fairly expressive guess.

4.1 Learning a Stopping Condition.

Our experiments showed that, when using ERM to train the model, limiting BP to only two iterations achieves maximum accuracy. This was true for several random MRFs that we generated using our procedure. Therefore, we did not experiment with a dynamic stopping condition on this synthetic dataset.

4.2 Explicitly Achieving Speed through Sparsity.

Finally, we experiment with our extension that learns an accurate sparse graph. We employ the following learning procedure: we first learn a model to optimize MSE without including the sparsity term. Starting with this model, we then learn a model including the sparsity term.

Figure 1 shows the speed vs accuracy curve for our method as compared to a models trained through ERM, but using L1-regularization to induce sparsity. In the former case we achieve different levels of sparsity (speed) by varying the parameter λ, which controls how much weight is given to the speed term in the loss function. In the case of L1-regularization, different levels of sparsity are achieved by varying the strength of the regularizer.

Our method achieves better accuracy of all levels of speed, with the exception of λ being close to 0 when no edges are excluded from the model.

5 Conclusions and Future Work

We showed that ERM is a good way to learn models when runtime is a consideration. We proposed two ways to speed-up computation: by including a dynamic stopping condition and by directly controlling the model sparsity. Experiments with the latter show that we are able to learn much sparser (and quicker) models at a modest accuracy cost.

We plan to also explore learning sparse models by gradually adding edges, as in [3] but using our speed-augmented objective. We also intend to devise methods for learning prioritization and pruning heuristics for message updates.

References

[1] G. Elidan, I. McGraw, and D. Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI)*, 2006.

[2] A. Griewank and G. Corliss. *Automatic differentiation of algorithms: theory, implementation, and application*. Society for Industrial and Applied Mathematics, 1991.

[3] Su-In Lee, Varun Ganapathi, and Daphne Koller. Efficient structure learning of Markov networks using L_1-regularization. In *Proceedings of NIPS*, pages 817–824, 2006.

[4] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[5] N.N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Proceedings of ANN*, pages 569–574, 1999.

[6] V. Stoyanov, A. Ropson, and J. Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, 2011.

[7] C. Sutton and A. McCallum. Piecewise training of undirected models. In *Proceedings of UAI*, pages 568–575, 2005.

[8] S.V.N. Vishwanathan, N.N. Schraudolph, M.W. Schmidt, and K.P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of ICML*, pages 969–976, 2006.

[9] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Bethe free energy, Kikuchi approximations and belief propagation algorithms. Technical Report TR2001-16, Mitsubishi Electric Research Laboratories, 2000.