General Purpose MRF Learning with Neural Network Potentials

Hao Xiong and **Nicholas Ruozzi**
The University of Texas at Dallas
{hao.xiong, nicholas.ruozzi}@utdallas.edu

Abstract

Maximum likelihood learning is a well-studied approach for fitting discrete Markov random fields (MRFs) to data. However, general purpose maximum likelihood estimation for fitting MRFs with continuous variables have only been studied in much more limited settings. In this work, we propose a generic MLE estimation procedure for MRFs whose potential functions are modeled by neural networks. To make learning effective in practice, we show how to leverage a highly parallelizable variational inference method that can easily fit into popular machining learning frameworks like TensorFlow. We demonstrate experimentally that our approach is capable of effectively modeling the data distributions of a variety of real data sets and that it can compete effectively with other common methods on multilabel classification and generative modeling tasks.

1 Introduction

Maximum likelihood estimation (MLE) is a standard approach for fitting Markov random fields (MRFs) to data. Much of the existing work on general MLE procedures for MRFs has focused on either discrete models or models with continuous but severely restricted potential functions, e.g., Gaussian graphical models [Malioutov *et al.*, 2006; Weiss and Freeman, 2001; Moallemi and Van Roy, 2009; Ruozzi and Tatikonda, 2013]. In these settings, MLE is often implemented as a double-loop procedure with an outer loop that performs gradient ascent over the model parameters while, in an inner loop, the gradients are estimated using a marginal inference procedure. As exact inference is, in general, NP-hard, the gradient is often approximated by sampling, message-passing algorithms such as belief propagation, or variational methods. For efficiency, these inner-loop approximate inference schemes are often not run to convergence, which can make the outer loop of the MLE procedure noisy and slow to converge in practice. Despite this, these approaches appear to perform well in practice Domke [2013]. Additionally, a variety of approaches that attempt to mitigate these issues for *discrete* MRFs have been proposed [Ganapathi *et al.*, 2008; Hazan and Urtasun, 2010; Domke, 2013; Tang *et al.*, 2016].

In this work, we are concerned with fitting parameterized MRFs, via standard double-loop MLE, to data sets containing both *continuous and discrete values*. There are two difficulties that arise in the continuous case that are not present in the discrete case. The first is a modeling problem: one needs to find a family of parameterized potential functions for continuous MRFs that is expressive enough to capture real continuous data but is also simple enough to make (approximate) learning feasible. The second issue is that inference in the continuous case is significantly more complicated than in the discrete case; standard methods such as belief propagation (BP) require either particle or non-parametric approximations [Sudderth *et al.*, 2003; Ihler and McAllester, 2009; Lienart *et al.*, 2015; Noorshams and Wainwright, 2013; Song *et al.*, 2011]. Particle message-passing algorithms can be slow to converge and are often unstable, which can make them impractical for the inner loop of MLE (especially in large models). Additionally, for belief propagation based methods there is no guarantee that the resulting marginals arise from some global joint distribution. This limits the utility of the learned beliefs, e.g., they cannot be used to generate new samples, and a separate sampling method is required in practice.

The primary contribution of this work is a generic parameter-based learning framework for MLE in MRFs with continuous variables. Our approach combines the flexibility of neural networks as function approximators with probabilistic models. Many different approaches that combine these two frameworks have been proposed, e.g., joint training of a Gaussian conditional random field (CRF) and a NN where the conditional features are given by the output of the NN [Do and Artieres, 2010], joint training of a CRF and a CNN by embedding the CRF into a recurrent neural network [Zheng *et al.*, 2015; Chen *et al.*, 2015], adding gate functions to emulate feature maps in discrete CRFs [Peng *et al.*, 2009], among others. In most applications of hybrid NN/CRF methods, the CRF is limited to discrete or simple Gaussian graphical models, often with mean-field variational inference, as approximate inference and learning in generic CRFs with arbitrary continuous potentials presents a more formidable challenge.

Here, in contrast to existing work, we propose to embed NNs directly into MRFs: We will model the potential functions of the MRF as NNs. We will show that a straightforward variational approximation combined with quadrature methods (or sampling) and iteration averaging can be successfully applied to fit such models to real data distributions. Our contributions include (1) a generic MLE procedure for MRFs with NN potential functions, (2) an averaging procedure that reduces the variance of the approximate inference procedure, which aids convergence and improves accuracy, and (3) a scalable implementation of this approach that can take advantage of modern GPU hardware. We demonstrate that our proposed approach performs competitively against standard methods on a variety classification and modeling tasks over continuous data sets.

2 Preliminaries

A Markov random field (MRF) over random vectors $x \in \mathbb{R}^n$ is defined by a hypergraph $G = (V, \mathcal{A})$ and a collection of non-negative potential functions such that the joint distribution p factorizes over the graph G, that is,

$$p(x) = \frac{1}{\mathcal{Z}} \exp\Big(\sum_{c\in\mathcal{A}} f_c(x_c)\Big), \tag{1}$$

where for each $c \in \mathcal{A}$, $f_c : \mathbb{R}^{|c|} \to \mathbb{R}$ is a function over x_c, the elements of x indexed by the hyperedge c, and $\mathcal{Z}$, often called the partition function, is the normalizing constant for the distribution.

As we will be interested in fitting models of this type to data, we will assume that the distribution p is parameterized by a vector $\theta \in \mathbb{R}^m$. Given data points $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(M)} \in \mathbb{R}^n$, we will attempt to find a setting of the parameters θ that maximizes the average log-likelihood:

$$\ell(\theta) \triangleq \frac{1}{M}\sum_m \Big[\sum_c f_c(\boldsymbol{x_c^{(m)}}|\theta) - \log\mathcal{Z}(\theta)\Big]. \tag{2}$$

To apply the standard double-loop MLE procedure, we compute the gradient of the log-likelihood with respect to θ.

$$\nabla\ell \triangleq \frac{1}{M}\sum_m\sum_{c\in\mathcal{A}} \Big[\nabla f_c(\boldsymbol{x_c^{(m)}}|\theta) - \mathbb{E}_{\boldsymbol{p}_c}\nabla f_c(\cdot|\theta)\Big], \tag{3}$$

where p_c is the marginal distribution over x_c. As a result, computing the gradient requires evaluating expectations with respect to the marginal distributions $\boldsymbol{p_c}(\cdot|\theta)$.

2.1 The Bethe Free Energy

Computing the marginal distributions of a given MRF requires performing statistical inference, an intractable operation for general potential functions and graphs. As a result, approximate inference or sampling methods are often used to estimate the gradient of the log-likelihood in practice.

Many different algorithms have been proposed for approximate inference in the continuous case: variational mean field, expectation propagation [Minka, 2001; Heskes and Zoeter, 2002; Heskes and Zoeter, 2005], nonparametric belief propagation [Sudderth *et al.*, 2003], particle belief propagation [Ihler and McAllester, 2009], expectation particle belief propagation [Lienart *et al.*, 2015], kernel belief propagation [Song *et al.*, 2011], stochastic orthogonal series message-passing [Noorshams and Wainwright, 2013], among others. Unfortunately, many of these methods are not efficient in practice [Guo *et al.*, 2019], and some of them place additional restrictions on the model, e.g., requiring each of the potential functions to be normalizable. In addition, the beliefs produced by message-passing algorithms do not arise as the marginals of a joint distribution, except for tree-structured models. This means that, in practice, Gibbs sampling or a similar method would need to be used to generate samples from the learned model, which is undesirable as the approximations made by Gibbs sampling do not align with those made while learning.

Instead, we will adopt a variational approach, similar to [Guo *et al.*, 2019; Xiong *et al.*, 2019; Gershman *et al.*, 2012], that approximates the marginals of p using Gaussian mixtures. Specifically, we use the Bethe free energy (BFE) approximation combined with Gaussian quadrature integral approximations. The BFE corresponds to a variational inference scheme that is exact on tree-structured models and is closely related to the belief propagation algorithm [Yedidia *et al.*, 2005]. For a pairwise MRF the BFE objective is given by

$$\mathcal{F}(b) = -\sum_{i\in\mathcal{V}}\mathbb{E}_{b_i}[f_i(x_i|\theta)] - \sum_{(i,j)\in\mathcal{E}}\mathbb{E}_{b_{ij}}[f_{ij}(x_i,x_j|\theta)] + \sum_{i\in\mathcal{V}}\mathbb{E}_{b_i}[\log b_i] + \sum_{(i,j)\in\mathcal{E}}\mathbb{E}_{b_{ij}}\left[\log\frac{b_{ij}}{b_i b_j}\right]. \tag{4}$$

Following [Guo *et al.*, 2019], we approximate the log-partition function, $\log\mathcal{Z}$, the univariate marginals, and the pairwise marginals by attempting to minimize the Bethe free energy over beliefs that are represented as fully factorized Gaussian mixtures, i.e., for all $i \in V$

$$\forall i \in V, b_i(x_i) \triangleq \sum_k \alpha_k b_i^k(x_i|\mu_i^k, \sigma_i^k)$$

and for all $(i,j) \in E$,

$$b_{ij}(x_i,x_j) \triangleq \sum_k \alpha_k b_i^k(x_i|\mu_i^k,\sigma_i^k) b_j^k(x_j|\mu_j^k,\sigma_j^k),$$

where each b_i^k is a normal distribution with mean μ_i^k and varianace σ_i^k and $\alpha_k \in [0,1]$ are the mixture weights.

With the above definitions, each belief b_i is a marginal distribution of the joint mixture $\sum_k \alpha_k \prod_{i\in V} b_i^k(x_i)$. The existence of such a joint distribution is in contrast to belief propagation based methods, which can produce beliefs that are not the marginals of any joint distribution. We will exploit the existence of this joint distribution as part of an averaging scheme to reduce the randomness inherent to the approximate inference process. Further, this distribution can be used as a surrogate for the learned model.

3 Neural Network MRFs

We propose to model the MRF potential functions using NNs. The advantage of this representation is that such potential functions can be made arbitrarily expressive, and we can control the expressivity of the representation by changing the structure, depth, and activation functions of the chosen NNs. One drawback of this approach is that variational methods,

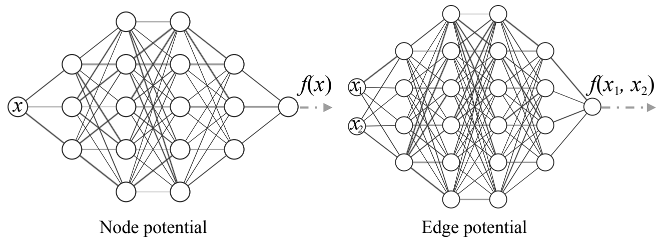

Figure 1: Example structures of neural networks for node and edge potential functions.

or indeed most of the above mentioned message-passing algorithms will need to sample from or compute expectations of functions described by general NNs. Many of the above inference methods cannot be applied in this setting without modification, e.g., if the method requires the potential functions to be normalizable.

Each node and edge potential of the MRF will be parameterized by the weights and biases of the corresponding NN. We will refer to these parameters as the model parameters or model weights θ. Specifically, each node potential is a single input neural network $f(x_n)$ and each edge potential is a double input neural network $f(x_1, x_2)$. All the networks are dense (fully connected) and have only one output value. See Figure 1. These small dense neural networks may not need to be very deep: NNs with only a few layers seem to to work well in practice. The small size helps to reduce the computational overhead, which makes the approach practical even for relatively large models, and can help combat overfitting.

Unlike exponential family models, the log-likelihood of our model is not convex in the model parameters, θ, as the individual NNs are non-convex functions of their weights. Nevertheless, we can still efficiently approximate the gradient of the log-likelihood with respect to θ and use gradient-based optimization to obtain a local optimum, hoping that such a local optimum will perform well in practice. In our implementation of the BFE minimization problem, we use well-studied Gaussian quadrature methods to approximate the expectations, which approximate the integrals as a weighted sum of a finite number of terms determined by a collection of quadrature points [Golub and Welsch, 1969], but sampling based methods, which could be used as part of a stochastic gradient descent procedure, also appear to perform well in practice. The inference procedure performs gradient descent by taking derivatives of (4) using the Gaussian quadrature method to approximate the expectations, with a time complexity of $O(NLK^2)$, where N is the total number of nodes and edges, L is the number of mixture components and K is the number of quadrature points. Once the inference task is complete, we obtain the gradient of the potential functions with respect to θ by back-propagation and then again apply Gaussian quadrature methods for the expectations in (3) to obtain the gradient of the log-likelihood.

As discussed in the introduction, double-loop MLE methods can become unstable when approximate inference is used in the inner loop. An explanation for why this might occur in our setting is that the BFE may have many different local optima, especially when the model is highly multimodal. As a result, even with θ fixed, repeated runs of gradient descent on (4) can yield very different beliefs. To combat the potential high variance of repeated inference, we propose a belief averaging procedure. Specifically, we use a belief pool to store the marginal beliefs produced during the inference procedure during the last T outer loop iterations. Instead of only using current round of $\boldsymbol{b_t}$ in the gradient computation of the log-likelihood, we use an average distribution $\bar{\boldsymbol{b}}_{\boldsymbol{T}}(x) \triangleq \frac{1}{T}\sum_t \boldsymbol{b_t}(x)$, which is a uniform mixture of all mixture models in the pool. Note that this is only possible because the variational inference process applied here returns a joint model: if we had used one of the message-passing algorithms instead, a different averaging operation would need to be considered. Note that averaging procedures have been shown to greatly reduce the noisiness of gradient methods, e.g., for averaging in the Frank-Wolfe algorithm see [Lacoste-Julien *et al.*, 2013]. We observe similar results experimentally here. The entire MLE process is summarized in Algorithm 1. For a high level visualization of the learning procedure, see Figure 2.

One technical note: regardless of how the potential functions are chosen, we must ensure that the resulting distribution is normalizable, i.e., $\mathcal{Z}$ exists and is finite. For example, if the output activation is tanh, which is a bounded function, this can be achieved by augmenting the univariate potential functions to be of the form $\exp(f_i(x_i|\theta) - cx^2)$ where $c > 0$ is a small positive number. This idea can be extended to a variety of different NN architectures.

Proposition 1. *If all of the potential functions are defined in terms of ReLU neural networks, augmenting the univariate potential functions as above, i.e.,* $\exp(f_i(x_i|\theta) - cx^2)$*, ensures that the resulting distribution is normalizable.*

Proof. (*Sketch*) A fixed ReLU neural network defines a piecewise linear function. The expectation of any log-linear function w.r.t. a Gaussian distribution exists and is finite. As a result, the expectation of any piecewise log-linear function (with finite pieces) w.r.t. a Guassian distribution exists and is finite. □

Finally, in order to make our implementation particularly efficient, we vectorize the inner most loop of Algorithm 1 into a single tensor operation by viewing quadrature points as broadcasted "data" on each clique, so that the gradients of the potential functions with respect to θ can all be evaluated in one batch using back-propagation. The marginal inference procedure outlined above is also embarrassingly parallelizable. As a result, the entire training process can be implemented quite efficiently on modern GPU accelerators.

4 Experimental Results

In this section we will illustrate the practical performance of our method. In particular, we will be interested in answering the following questions.

- How well does the learned model approximate the true data generating distribution?
- How accurate is the learned model for classification?
- Does the learned model generalize well to unseen data?

Algorithm 1: MLE with NN potentials

```
1  Randomly initialize weights $\theta$ for NN potential functions $f$;
2  Initialize marginal belief parameters for each mixture
     component $k$, $(\alpha_k, \mu^k_{i\in V}, \sigma^k_{i\in V})$;
3  $t = 0$, $\eta = 1$;
4  Initialize belief queue $Q_T$ of size $T$ with $\boldsymbol{b_t}$;
5  foreach epoch do
6      Shuffle training dataset;
7      foreach batch data of data, B do
8          Compute $\boldsymbol{b_t}$ via gradient descent on (4);
9          PUSH $\boldsymbol{b_t}$ into $Q_T$, POP the oldest out;
10         $\bar{\boldsymbol{b_t}} = \frac{1}{T}\sum_t \boldsymbol{b_t}$ for each $\boldsymbol{b_t}$ in $Q_T$;
11         foreach training example $m$ in B do
12             $\nabla\theta_{mt} =$
                 $\sum_{c\in\mathcal{A}}\left[\nabla f_c(\boldsymbol{x}_c^{(m)}|\theta) - \mathbb{E}_{\bar{\boldsymbol{b_t}}}\nabla f_c(\cdot|\theta)\right]$
13         end
14         $\theta_{t+1} = \theta_t + \eta \cdot \frac{1}{|B|}\sum_{m\in B}\nabla\theta_{mt}$;
15         $t = t + 1$;
16         $\eta = \eta \cdot decay\_rate$;
17     end
18 end
```

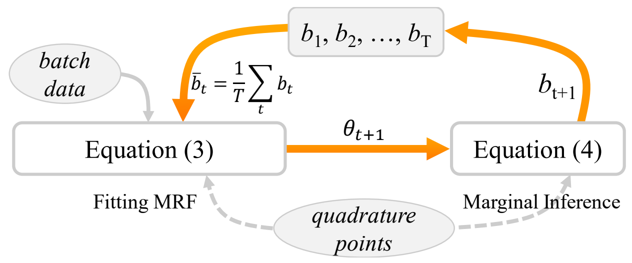

Figure 2: Data flow model for Algorithm 1. The gray nodes represent computations carried out in TensorFlow. The *quadrature points* are selected by the Gaussian quadrature method for the expectations in the marginal inference procedure and in the gradient computation.

As our aim is to demonstrate that this is a generic, widely applicable tool, we compare against standard methods for classification and model fitting: SVMs, logistic regression, decision trees, Gaussian naïve Bayes, etc.

Our algorithm was implemented in Python using the TensorFlow 2 framework to take advantage of fast GPU training and inference. For the competing methods, we used the implementations available in scikit-learn. All code and data used as part of these experiments will be made available on GitHub.[1]

4.1 MRF Learning on Synthetic Data

We begin with a synthetic experiment in which we train a model using data sampled from a known distribution and compare the learned distribution with the true distribution. The true distribution is chosen to be a mixture of two trivariate Gaussian distributions, and the means and covariance matrices Σ were randomly generated for each of the 10 runs. Our aim is to learn a pairwise MRF with NN potential functions that factorizes over the 3-cycle to data generated from this mixture model.

[1]https://github.com/motionlife/nnmrf

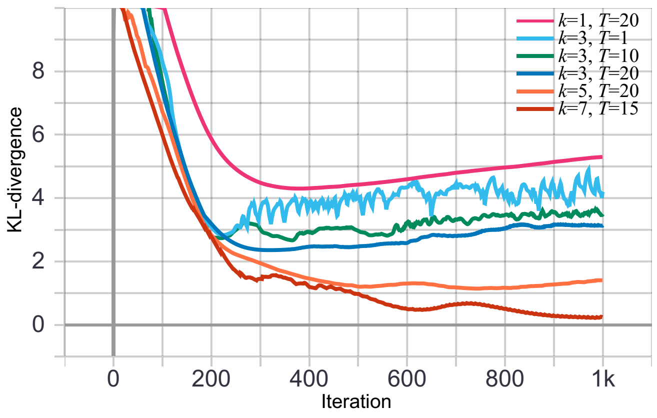

Figure 3: KLD between the learned distribution with k mixture components and pool size T and the true distribution for the synthetic Gaussian mixture model experiment.

In each run of this experiment, we sampled 2000 data points from the mixture and applied the learning procedure in Algorithm 1. Both node and edge potentials are neural nets with 4 hidden layers and the hidden nodes for each layer are $[3, 5, 5, 3]$ for each node potential, $[5, 7, 7, 5]$ for each edge potential respectively, with all hidden layer's activation function set to be SeLUs (scaled exponential linear units) [Klambauer *et al.*, 2017] and the output layer activation is TanH.

To assess model fit, we compute the KL-divergence (KLD) between the learned model $\tilde{p}$ and the ground truth model p: the KLD is always non-negative, and smaller KLD corresponds to a better model fit. Figure 3 shows KLD versus iteration number for varying numbers of mixture components, k, and different sizes for the averaging set, T. As expected, KLD decreases as k increases: if the number of mixture components is too small, the approximate inference procedure is likely to become stuck in poor local optima, which results in a poor approximation of the marginals for the gradient update. Note that while the approximate marginals are mixtures of Gaussian distributions with diagonal covariance matrices, the true distribution has a full covariance matrix. As a result, multiple mixture components in the fully factorized case are needed to approximate a single component of p.

In addition, we find that increasing the pool size T for the averaging technique significantly stabilizes the training process and also appears to encourage the learning to converge to a better distribution. The first observation here is not so surprising as averaging reduces variance. For the latter observation, our intuition is that as T increases, the average model, which is a mixture of all of the local optima found during the inference procedure, is likely to contain an increasing number of the possible local optima. As a result, even if the inference procedure fails to find the globally optimal mixture on any single iteration, it will likely find different pieces of this mixture. These different pieces tend to correspond to peaks of the true distribution. And, as a result, the average model is likely to be a mixture of all of the most likely peaks of the data generating distribution.

Though the underlying model was quite simple, this experiment provides some evidence that it is possible for the NN-MRFs produced using Algorithm 1 to result in a reasonable model fit in a multimodal setting.

4.2 Discriminative Task: Classification

In a second experiment, we evaluated the training and test accuracy of our method (NN-MRF) on eight data sets from the UCI repository and compare the results with a variety of standard classifiers: Support Vector Machines (SVMs), Gaussian Naive Bayes (GNB), logistic regression (LR) , decision trees (DTs), random forrests (RFs), k-nearest neighbor (KNN), and multi-layer perceptrons (MLP). The number of features in these data sets ranged from 4 to 60 and are all treated as continuous values. The data set sizes range from 150 to 10,000.

For simplicity, we chose the complete graph with pairwise potential functions for the MRF representation. To use our model as a classifier for label y, we simply train a different MRF for each conditional distribution $p(x|y)$ for each y simultaneously. The joint distribution is then given by $p(y)p(x|y)$, and classification is performed for an unseen x by maximizing $p(y|x) \propto p(y)p(x|y)$ over all possible values of the class label y. Note that $p(y)$ is also estimated via MLE.

In this experiment, the node potential functions are fully connected SELU NNs with 4 hidden layers $[\,3,\ 4,\ 4,\ 3\,]$ and the edge potentials are also fully connected SELU NNs with 4 hidden layers $[\,4,\ 5,\ 5,\ 4\,]$. The output layer activation is tanh. We used a belief pool size of $T = 20$, and we ran 100 iterations in each marginal inference procedure during learning with the ADAM optimizer. For the MLP classifier, we used 3 hidden layers with each layer's node number based on the corresponding dataset's feature number n, i.e., $[\,3n,\ 5n,\ 3n\,]$ and increased the maximum number of iterations to 500. SVMs were applied using Gaussian kernels with penalty parameter 2. We also increased the number of trees for the Random Forrest to 150 and the number of neighbors used by KNN to 10 with its leaf size set to 50. All other parameters were kept at the Python scikit-learn package defaults.

Table 1 shows the average test accuracy of all methods over 30 randomly chosen 80/20, train/test splits. From the table, we can see that NN-MRF and MLP are consistently among the top performers on each data set. Particularly we find that NN-MRF can always fit the training data very well, which suggests that the model is very expressive, and its similar performance on testing data suggests that the model is not severely overfit to the training data. Note that overfitting can easily occur when using MLE on continuous data.

4.3 Comparing Generative Models

In addition to performance on the classification task described in the last section, the quality of a generative model can also be evaluated based on how well the learned distribution matches the data distribution. Here we assess the performance of NN-MRFs for the model fitting task.

Test Set Log-Likelihood

One way to assess the quality of a model fit is to look at the probability of generating the test data under the model.

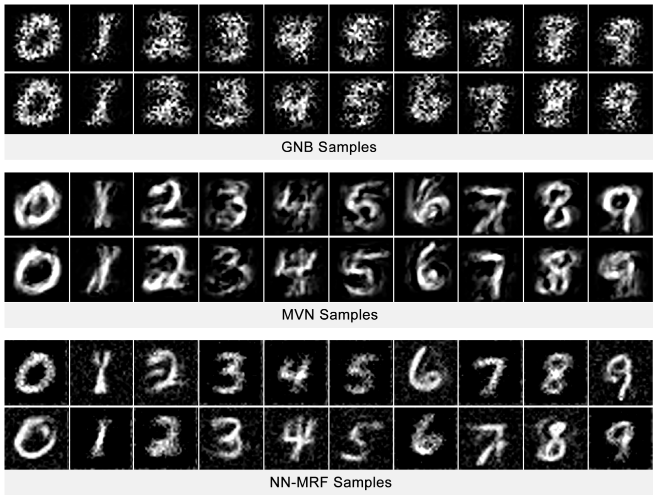

Figure 4: Sample images generated by GNB (top), MVN (middle), and NN-MRF (bottom).

Among all the classifiers fit in the last subsection only NN-MRF and GNB yield models of the data generating distribution. Here, we compare these two models with a third approach, dubbed multivariate normal (MVN), in which we fit a separate Gaussian distribution for each class label.

Table 2 shows the average test set log-likelihood for the models fit to both the UCI datasets from the previous section as well as the MNIST image classification data set [LeCun *et al.*, 1998] using the above three methods. In terms of test set log-likelihood, NN-MRF is much better than GNB nearly every data set, even in cases when the error of these models as classifiers is comparable. This is not entirely surprising as GNB only fits a single, fully factorized Gaussian dsitribution for each class label while NN-MRFs fit an entire MRF with NN potentials per class label. The same is true for the comparison between NN-MRF and MVN: NN-MRF produces a better test set log-likelihood on all data sets except for one, the UCI Leaf data set. Given that NN-MRF achieved its worst training performance on this data set, and that the fit is better with a unimodal distribution, it suggests that more mixture components are likely needed in order for the NN-MRF to better capture the variance data generating distribution.

On MNIST, NN-MRF achieves 96.02% test accuracy with a grid MRF for each class label and four mixture components, while GNB's performance is only about 84.6%. Although it has the same μ vector as GNB, MVN has a test accuracy of 95.42%. Both NN-MRF and MVN both outperform GNB in terms of test log-likelihood. As MNIST is an image data set we can visualize the performance of the methods. Figure 4 shows samples generated by each of the methods.Although NN-MRF narrowly outperforms MVN in terms of model fit and test accuracy, the samples suggest that the models are quite different practically.

While $> 96\%$ is a reasonable score on this classification task, convolutional neural networks (CNNs) can achieve accuracy $> 99\%$. To achieve this level of performance with our approach it is likely that we may need to change the potential functions, e.g., to CNNs, or the MRF. Additionally, latent

	Train	Test							
Dataset	**NN-MRF**	**NN-MRF**	**SVM**	**GNB**	**LR**	**DT**	**RF**	**KNN**	**MLP**
Iris	0.9902	0.9667	0.9633	0.9500	0.9767	0.9166	0.9600	0.9533	**0.9833**
Seeds	0.9735	**0.9778**	0.9048	0.8881	0.9238	0.9119	0.9142	0.9071	0.9381
Sonar	0.9945	0.7789	0.6889	0.6711	0.7333	0.6977	**0.7844**	0.7778	0.7822
Banknote	0.9969	**1.000**	**1.000**	0.8423	0.9905	0.9819	0.9914	0.9967	**1.000**
Leaf	0.852	0.7935	0.6382	0.5985	0.6044	0.6926	0.7661	0.7058	**0.8074**
Glass	0.9511	0.7895	0.7429	0.6057	0.6657	0.7142	**0.8085**	0.7628	0.7429
Electricity	0.9994	**1.000**	0.8921	0.9748	0.8831	0.9993	0.9993	0.7734	0.9731
Protein	0.9871	**0.9837**	0.9793	0.7345	0.9431	0.9379	0.9568	0.9689	0.9586

Table 1: Classification accuracy of different methods on various UCI data sets. The best average performance appears in bold.

Dataset	NN-MRF	GNB	MVN
Iris	**4.235**	−2.353	-1.102
Seeds	**5.632**	−1.859	5.160
Sonar	**188.037**	61.911	124.132
Banknote	**−4.357**	−10.891	-9.159
Leaf	21.019	15.240	**27.175**
Glass	**8.941**	0.190	6.801
Electricity	**−3.783**	−11.042	-4.169
Protein	**14.587**	6.424	12.907
MNIST	**281.195**	−1451.464	260.884

Table 2: Comparison of the average test set log-likelihood for NN-MRFs, GNB, and MVN over various UCI data sets.

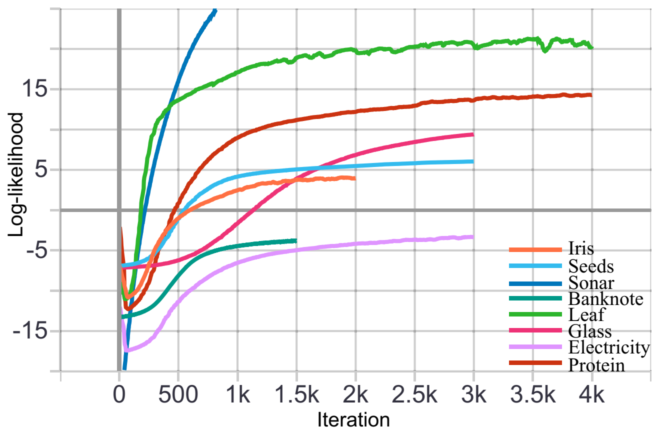

Figure 5: Average test set log-likelihood versus iteration for NN-MRF on various UCI data sets.

variables could capture the same kind of convolution structure that seems to work well on image classification tasks with deep NNs. We leave these extensions for future work.

To better understand the rate of convergence of the learning process for NN-MRFs, we plot the test set log-likelihood per iteration in Figure 5. While convergence rate greatly depends on the model parameters' learning rate and the number of mixture components used by the model, the overall trend is that after a few thousand iterations, the test set log-likelihood is relatively stable.

Visual Inspection

To provide further evidence of the quality of models fit by NN-MRFs, we use the t-distributed stochastic neighbor embedding (t-SNE) technique to visualize our multidimensional data. t-SNE is a nonlinear dimensionality reduction technique that projects high dimensional data instances into a 2-D or 3-D space such that points that are more similar in the high dimensional space are likely to be closer in the projected space.

We sampled data from our trained probabilistic model $\tilde{p}(x|y) = \sum_k \alpha_k \prod_i b_i^k(x_i)$ and projected it along with the original data set using t-SNE. Figure 6 visualizes the UCI seeds data set, with 7 features and 3 class labels (Kama, Rosa and Canadian), in two dimensions with the perplexity set to 30 under Euclidean distance. The sampled data set size was chosen to be the same as the original data set size. As we can see, the sampled data from our model matches quite well with the original data set, which provides further evidence that NN-MRFs produce reasonably good approximations to the data generating distribution.

5 Discussion

Fitting expressive MRF models to continuous data is a highly non-trivial problem that requires both an efficient inference algorithm and an appropriate parametric family for the potential functions. For example, this work was motivated in part by an initial investigation into fitting MRFs with log-polynomial potential functions as, theoretically, the Gaussian quadrature integral approximation methods can accurately compute the expectations in the BFE in this case with a small number of quadrature points. Experimentally, however, we found that it was extremely difficult to achieve convergence of the learning procedure for these models, and when it did converge, the resulting model was often worse than GNB.

We presented a general methodology for parametric MLE for MRFs over continuous variables that can be implemented efficiently using modern GPUs. We showed experimentally that the learning procedure results in accurate models of the data generating distribution as well as an accurate and efficient classification procedure. These results suggest that this method is a good candidate for inclusion into the standard toolbox of widely applicable machine learning methods. In addition, the method can easily handle hybrid MRFs, i.e., graphical models with both discrete and continuous random variables, by allowing the mixture components to be fully factorized products of arbitrary univariate distributions.

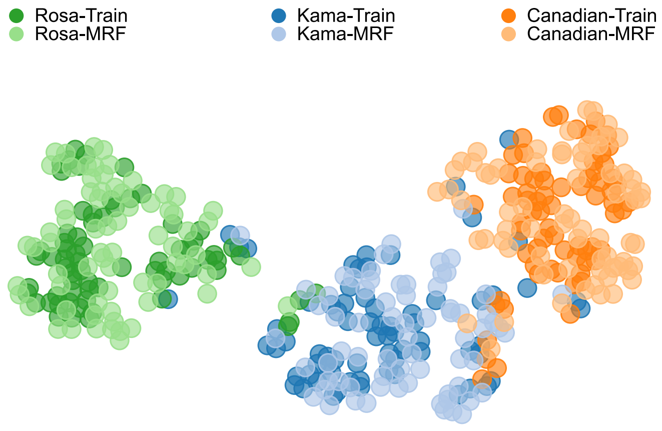

Figure 6: t-SNE visualization for seeds data and samples generated by the learned NN-MRF model. Best viewed in color.

Acknowledgements

This work was supported, in part, by the DARPA Explainable Artificial Intelligence (XAI) program (N66001-17-2-4032). Any opinions, findings, or recommendations are those of the authors and do not necessarily reflect the view of the DARPA or the US government.

References

[Chen *et al.*, 2015] L.-C. Chen, A. Schwing, A. Yuille, and R. Urtasun. Learning deep structured models. In *International Conference on Machine Learning (ICML)*, 2015.

[Do and Artieres, 2010] T. Do and T. Artieres. Neural conditional random fields. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[Domke, 2013] J. Domke. Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2454–2467, 2013.

[Ganapathi *et al.*, 2008] V. Ganapathi, D. Vickrey, J. Duchi, and D. Koller. Constrained approximate maximum entropy learning of Markov random fields. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008.

[Gershman *et al.*, 2012] S. J. Gershman, M. D. Hoffman, and D. M. Blei. Nonparametric variational inference. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 235–242, 2012.

[Golub and Welsch, 1969] G. H. Golub and J. H. Welsch. Calculation of Gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

[Guo *et al.*, 2019] Y. Guo, H. Xiong, and N. Ruozzi. Marginal inference in continuous Markov random fields using mixtures. In *33rd Conference on Artificial Intelligence (AAAI)*, Jan. 2019.

[Hazan and Urtasun, 2010] T. Hazan and R. Urtasun. A primal-dual message-passing algorithm for approximated large scale structured prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2010.

[Heskes and Zoeter, 2002] T. Heskes and O. Zoeter. Expectation propagation for approximate inference in dynamic bayesian networks. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 216–223, 2002.

[Heskes and Zoeter, 2005] T. Heskes and O. Zoeter. Gaussian quadrature based expectation propagation. In *Proceedings of the Eighth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2005.

[Ihler and McAllester, 2009] A. T. Ihler and D. A. McAllester. Particle belief propagation. In *Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 256–263, 2009.

[Klambauer *et al.*, 2017] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks, 2017.

[Lacoste-Julien *et al.*, 2013] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank-Wolfe optimization for structural svms. In *International Conference on Machine Learning (ICML)*, 2013.

[LeCun *et al.*, 1998] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Lienart *et al.*, 2015] T. Lienart, Y. W. Teh, and A. Doucet. Expectation particle belief propagation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3609–3617, 2015.

[Malioutov *et al.*, 2006] D. M. Malioutov, J. K. Johnson, and A. S. Willsky. Walk-sums and belief propagation in Gaussian graphical models. *Journal of Machine Learning Research*, 7:2031–2064, 2006.

[Minka, 2001] T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in Artificial Intelligence (UAI)*, pages 362–369, 2001.

[Moallemi and Van Roy, 2009] C. C. Moallemi and B. Van Roy. Convergence of min-sum message passing for quadratic optimization. *Information Theory, IEEE Transactions on*, 55(5):2413 –2423, May 2009.

[Noorshams and Wainwright, 2013] N. Noorshams and M. J. Wainwright. Belief propagation for continuous state spaces: stochastic message-passing with quantitative guarantees. *Journal of Machine Learning Research (JMLR)*, 14(1):2799–2835, 2013.

[Peng *et al.*, 2009] J Peng, L Bo, and J Xu. Conditional neural fields. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2009.

[Ruozzi and Tatikonda, 2013] N. Ruozzi and S. Tatikonda. Message-passing algorithms for quadratic minimization. *The Journal of Machine Learning Research*, 14(1):2287–2314, 2013.

[Song *et al.*, 2011] L. Song, A. Gretton, D. Bickson, Y. Low, and C. Guestrin. Kernel belief propagation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 707–715, 2011.

[Sudderth *et al.*, 2003] E. B. Sudderth, A. T. Ihler, M. Isard, W. T. Freeman, and A. S. Willsky. Nonparametric belief propagation. In *Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society Conference on*, 2003.

[Tang *et al.*, 2016] K. Tang, N. Ruozzi, D. Belanger, and T. Jebara. Bethe learning of graphical models via MAP decoding. In *Artificial Intelligence and Statistics (AISTATS)*, 2016.

[Weiss and Freeman, 2001] Y. Weiss and W. T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Comput.*, 13(10):2173–2200, Oct. 2001.

[Xiong *et al.*, 2019] H. Xiong, Y. Guo, Y. Yang, and N. Ruozzi. One-shot marginal map inference in Markov random fields. In *Proc. 35th Uncertainty in Artifical Intelligence (UAI)*, Tel Aviv, Israel, July 2019.

[Yedidia *et al.*, 2005] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *Information Theory, IEEE Transactions on*, 51(7):2282 – 2312, July 2005.

[Zheng *et al.*, 2015] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.