This paper was submitted as a final project report for CS6424/ECE6424 *Probabilistic Graphical Models and Structured Prediction* in the spring semester of 2016.

The work presented here is done by students on short time constraints, so it may be preliminary or have inconclusive results. I encouraged the students to choose projects that complemented or integrated with their own research, so it is possible the project has continued since this report was submitted. If you are interested in the topic, I encourage you to contact the authors to see the latest developments on these ideas.

Bert Huang
Department of Computer Science
Virginia Tech

Inference with Deep Neural Networks

Bijaya Adhikari* Virginia Tech bijaya@cs.vt.edu

William Ryan Doan * Virginia Tech willrd9@vt.edu

Jason Granstedt * Virginia Tech eveneth@vt.edu

Abstract

Inference on a probabilistic graphical model (PGM) is a challenging task which has been shown to be NP-Complete. Even approximate inference on an arbitrary PGM is not trivial as approximate inference algorithms are not guaranteed to converge to the global optimum. This convergence could occur at a local optimum resulting in a sub-optimal solution. In this project, we determine that given offline time we can estimate approximate inference for every variable in a PGM in an O(1) time complexity.

Recent studies show that deep learning is well suited to estimate intractable problems. In this project, we propose training an end-to-end deep neural network to infer states of graphical models with a specified structure and perform estimates of approximate inference on graphical models with arbitrary factors in a single forward pass. By doing so, we aim to combine graphical models and deep learning which has been of much research interest in recent years.

1 Introduction

Probabilistic graphical models show a significant amount of promise in modeling the complex probability distributions seen in real-world problems. However, many of the current algorithms for calculating inference are infeasible for the datasets usually encountered. Even approximate inference within absolute or relative error is NP-hard on general graphical model [10]. However, advancements in the field of deep learning have shown significant improvements in formerly intractable problems. Combining the power deep neural networks with the rich class of distributions representable by PGMs has a great deal of potential. Previous integration of neural networks and graphical models has been done before, but it was limited to appending a PGM to the end of a deep network to model dependencies [1]. We propose to design and implement an end-to-end neural network that learns a particular graphical model and derives a more efficient inference function.

Our neural network takes the CPDs corresponding to a MRF of known structure as inputs and outputs an estimate of the network's marginals. Since CPDs are traditionally difficult to acquire, we also train another neural network to learn them for a given predefined structure. The neural network model takes some time to train, but when completed is able to calculate the inference for a given graphical model state in a single forward pass. The most expensive part of the operation is generating the CPDs necessary to train the neural network.

There are two noteworthy limitations to this approach. First, a graphical model must be constructed and inference passing performed to generate the necessary data for training the neural network. If we want exact inference updates to train our neural network, we will have to run exact inference on a graphical model. This is intractable for larger networks, so we consider a small graphical model first. Once we demonstrate that we are successful in modeling this graph, we introduce our method for extracting CPDs from an input and demonstrate our approach's effectiveness.

*Equal Contribution.

Probabilistic Graphical Models and Structured Prediction Final Project Report. Spring 2016, Virginia Tech.

The second limitation is that this neural network will not be able to be altered once trained. Although it can be expensive to add another node to a graphical network, it is possible. A neural network, on the other hand, is a static structure once constructed. Thus, if it becomes necessary to alter the graphical model once the neural network is trained, an entirely new corpus of data must be generated and the training process redone. This limitation may also apply to the inquiry types that the network is capable of, although future research may reveal that we are able to circumvent the incomplete observation issue by including missing data examples.

In summary, we present a method to generate a neural network that can answer an inference query for a graphical model much more quickly than standard inference algorithms. The tradeoffs are the inflexibility of the resulting model and possible inaccuracies if approximate inference methods must be used to generate the training data.

2 Preliminaries

Next we give a brief background. For simplicity we do our analysis on Markov Networks (undirected graphical models). However, our analysis can be easily extended to other graphical models as well.

Bayesian Networks are defined by set of random variables $\mathcal{X} = (X_1, X_2, ..., X_N)$ in a graphical model $\mathcal{G}$, and set of conditional probability distribution for each parent-child relation in $\mathcal{G}$. Probability distribution of $\mathcal{X}$ factorizes as

$$P(\mathcal{X}) = \prod_{i=1}^{N} P(x_i|\text{parents}(x_i) \text{ in } \mathcal{G}) \tag{1}$$

Markov Networks are defined by set of random variables $\mathcal{X} = (X_1, X_2, ..., X_N)$ in a graphical model $\mathcal{G}$, and set of factors $\Phi = \{\phi_1(D_1), \phi_2(D_2), ..., \phi_m(D_m)\}$ where $\forall_{i=1}^{M} D_i \subseteq \mathcal{X}$ and each D_i is a complete subgraph in $\mathcal{G}$. Probability distribution of $\mathcal{X}$ factorizes as

$$P_\Phi(\mathcal{X}) = \frac{1}{Z}\bar{P_\Phi}(\mathcal{X}) \tag{2}$$

where unnormaized measure $\bar{P_\Phi}$ is

$$\bar{P_\Phi} = \prod_{m=1}^{M} \phi(D_m)$$

and normalizing constant Z is

$$Z = \sum_{\mathcal{X}} \bar{P_\Phi}(\mathcal{X})$$

Marginal Inference in a Bayesian Networks and Markov Networks involves conditional probability queries such as $P(X_1, X_2, X_{N-1}|X_N = x_n)$. Marginal Inference is NP-hard as it can be reduced to 3-SAT problem, which is known to be NP-Complete. However, there are algorithms such as Variable Elimination and Belief Propagation which provide exact solution to marginal inference in special Bayesian Networks. In general, no known algorithms can efficiently solve exact inference problem.

Deep Learning is a branch of machine learning and a composition of artifical neural networks (ANN). ANNs accept a vector of features and perform non-linear operations on the input and attempt classification after these operations have been performed. This entails inputing data into many stages of non-linearities which track errors that are propogated back through the network in order to correctly assign weight variables between these layers that are used in predictive models. 'Deep' layers, or many non-linear layers, are used in order to effectively capture data complexities that other simpler models may not be able to.

3 Problem Formulation and Methodology

In this project, we are interested in solving marginal inference problems by training deep neural networks. Let $\mathcal{X} = (X_1, X_2, ..., X_N)$ be set of variables in a graphical model $\mathcal{G}$. Then as previously

stated marginal inference queries are in form of $P(\mathcal{Y}|\mathcal{Z}=z)$, where both $\mathcal{Y}$ and $\mathcal{Z}$ are subset of $\mathcal{X}$. Our goal is to optimize the running time of answering such queries with increased efficiency given that we have enough offline time to train deep neural network. Next, we formalize our problem and propose our solutions.

3.1 Problem 1

Solutions to marginal inference queries are essentially a set of probability distributions which comprises unary distributions for each variable. Although distributions for factors are also calculated, they are not typically provided as answers to marginal inference queries. Given a probabilistic graphical model $\mathcal{G}$, set of variables $\mathcal{X}$, set of factors Φ and a marginal inference query q, let $Q_q(x)$ be the true marginals of variables in $x \in \mathcal{X}$ and $P_q(x)$ be inferred marginals. Then our problem can be formally stated as:

Problem 1. *For each $x \in \mathcal{X}$, given $\mathcal{G}$, q, Φ,and Q, find $P_q^*(x)$ such that*

$$P_q^*(x) = \arg\min_{P_q(x)} D(P_q(x), Q_q(x))$$

Where D is some measure of distance between probability distributions $P_q(x)$ and $Q_q(x)$. In this project, we use Hellinger's distance [5] to measure the distance between two distributions. Therefore, $D(P_q, Q_q)$ is defined as

$$D(P_q(x), Q_q(x)) = \frac{1}{\sqrt{2}} \left|\left| \sqrt{P_q(x)} - \sqrt{Q_q(x)} \right|\right|_2 \tag{3}$$

Now let S be set of states of variable x, So equivalently,

$$D(P_q(x), Q_q(x)) = \frac{1}{\sqrt{2}} \sqrt{\sum_{s \in S} \left(\sqrt{P_q(s)} - \sqrt{Q_q(s)} \right)^2} \tag{4}$$

To solve Problem 1, we propose training neural network via gradient descent. Once the network parameters w are learned for our deep network F, then inference amounts to single forward pass in F. i.e.

$$P_q(\mathcal{X}) = F(w, q, \Phi, \mathcal{G}) \tag{5}$$

To train a neural network we use squared error to minimize Equation 4. We provide justification for minimizing squared distance by showing that it is related to Hellinger's distance in the following lemma.

Lemma 1. *Minimizing the squared distance between $P_q(x)$ and $Q_q(x)$ for each $x \in \mathcal{X}$ minimizes Hellinger's distance between $P_q(x)$ and $Q_q(x)$.*

Proof. Let S be set of state variables $x \in \mathcal{X}$ can take on. Now, By definition, we have

$$\arg\min_{P_q(x)} D(P_q(x), Q_q(x)) = \arg\min_{P_q(x)} \frac{1}{\sqrt{2}} \sqrt{\sum_{s \in S} \left(\sqrt{P_q(s)} - \sqrt{Q_q(s)} \right)^2}$$

Removing the constant factor and noting that square root is monotonic for positive real numbers,

$$\arg\min_{P_q(x)} D(P_q(x), Q_q(x)) = \arg\min_{P_q(x)} \sum_{s \in S} \left(\sqrt{P_q(s)} - \sqrt{Q_q(s)} \right)^2$$

We note that $Q_q(s)$ is fixed for all $s \in S$ and $Q_q(x)$ satisfies the probability constraint $\sum_{s \in S} Q_q(s) = 1$ for each data . Therefore, states in $P_q(x)$ could be assumed to be independent for an optimization task. Therefore we get,

$$\arg\min_{P_q(x)} D(P_q(x), Q_q(x)) = \arg\min_{P_q(x)} \forall_{s \in S} \left(\sqrt{P_q(s)} - \sqrt{Q_q(s)} \right)^2 \tag{6}$$

□

The complete back propagation algorithm is presented in Algorithm 1.

Algorithm 1 Deep Inference

Require: Factor potentials Φ
while not converged **do**
  for every data point **do**
    Forward pass to compute $F(w, q, \Phi, \mathcal{G})$
    Calculate loss based on Equation 6.
    Backward pass via chain rule
    update w

3.2 Problem 2

We extend Problem 1 by reformulating it into a more general framework that does not require the CPDs as input, but rather that directly performs MAP inference given raw data. This problem has applications in image processing, natural language processing e.t.c.

To solve this problem, we propose training two independent deep networks. The first deep network takes the raw data as input and learns potentials (CPDs). Since we are assuming the Markov network structure is consistent for the inputs we are considering, we can specify a neural network that takes the raw input and learns the CPDs. This neural network is trained by running an inference algorithm over several different inputs to compute a list of CPDs. The input is treated as the training data, while the computed CPDs serve as the target. Choosing whether to use an exact or approximate inference algorithm for generating the training data will cause the same tradeoffs observed in a graphical model to apply to the generated neural network. Precisely, exact inference will produce the most accurate results but is intractable, while approximate inference algorithms will decrease the data generation time significantly but will result in a less accurate neural network.

The second deep network takes potentials of a Markov network as input and performs marginal inference. Then, it takes the most likely states for each variable as a MAP state. See the process for training neural networks to generate CPDs in Algorithm 2.

Once both deep networks are trained, MAP inference amounts to single forward pass via the first neural network to learn the CPDs and then a single forward pass via second neural network to learn the marginals and taking most likely states for each variable.

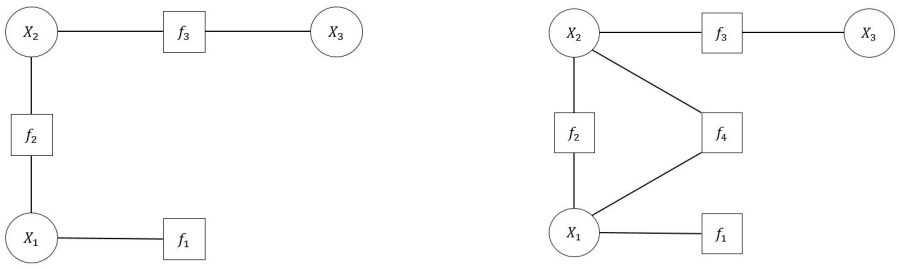
Algorithm 2 CPTrain

Require: Image data X
while not converged **do**
  for every data point $x \in X$ **do**
    Forward pass to compute $F(w, x)$
    Calculate squared loss.
    Backward pass via chain rule
    update w

4 Experiments

Our first experiment included both a proof of concept and a comparison between training the neural network with exact and approximate data. We chose to use the structure of a simple model found in Figure 1a. We generated CPDs randomly to create the necessary data and used belief propagation to compute the marginals. We then trained a neural network with the inputs of the factors representing each node. The training target of the network was the marginal probabilties obtained from belief propagation for each variable.

We then added an edge to the simple model, as shown in Figure 1b. We generated exact data for Figure 1a and approximate data using loopy belief propagation for Figure 1b . The results of these experiments are included in Table 1. While using exact inference results in a slightly more accurate model, it takes significantly more time to train. The results are also surprisingly accurate given that we are using a neural network with only a single hidden layer to perform inference.



(a) Non-loopy factor graph.

(b) Loopy factor graph

Figure 1: Factor graphs used for the experiments. The loopy factor graph was constructed from the original factor graph by placing an additional edge with a corresponding factor between X_1 and X_2. The second factor graph is loopy because f_2 and f_4 are not merged and have distinct values

Table 1: Results for the first experiment

Method Used	Accuracy	Execution Time (s)	Efficiency
BP-NonLoopy	**1.0**	.00332	301.2
NN-NonLoopy	.984	**.0018**	**546.6**
BP-Loopy	**1.0**	.00792	126.26
NN-Loopy BP Data	.955	**.0018**	**530.55**

We define efficiency here as the accuracy divided by the computation time. Accuracy is measured on a zero to one scale. As expected, our neural network approach sacrifices a small amount of accuracy for a large speedup in execution time.

Our second experiment used a more general problem formulation. We chose to perform an image denoising task on a subset of the MNIST dataset. The MNIST dataset is a collection of 70,000 handwritten images of numbers. Each image is comprised of 28 by 28 pixels, giving a total value of 784 pixels per image. The original dataset is in grayscale, with a range of 0 to 255 for each pixel. For our task, we converted the images to black and white to reduce the computation time needed for generating the training data. We chose the first 10,000 images of the dataset to be our training data for the neural network. We added uniform salt and pepper noise with a 5% chance to flip any pixel in the image.

We independently trained two neural networks for the image denoising task. The first neural network was trained to predict the CPDs of a pairwise Markov random field lattice graph of a noisy image. The second neural network was trained to predict the denoised image. Training for the second neural network was done by inputting the CPDs generated by the first neural network and training the output to achieve a target value of the denoised image. Both neural networks had a single hidden layer. A diagram of the entire system can be found in Figure 2.

We also trained an end-to-end neural network for the same task to compare our results to. This neural network was given the noisy image and trained to predict the clean image. To ensure a fair comparison, we specified this neural network with three hidden layers, each of the same size as the corresponding layer in the previous model. A diagram of this neural network can be found in Figure 3. The preliminary results of our experiment can be found in Figure 4. NN-Structure 1 refers to the structure detailed in Figure 2, while NN-Structure 2 refers to the structure detailed in Figure 3.

Firstly, the results strongly indicate that given offline time, our presented method is efficient. Our accuracies divided by the time required to perform online computations are significantly higher than classical methods consisting of belief propagation and inference of most likely configurations.

Secondly, the convergence graph shows that the error with respect to both stages of the piecewise neural network converge faster and more optimally with equal iterations compared to a single end-to-end neural network. This means that our method of producing CPDs from input vectors and then isolating the most likely configuration in a mirrored network surpasses the ability of a standalone network to accomplish the identical task.

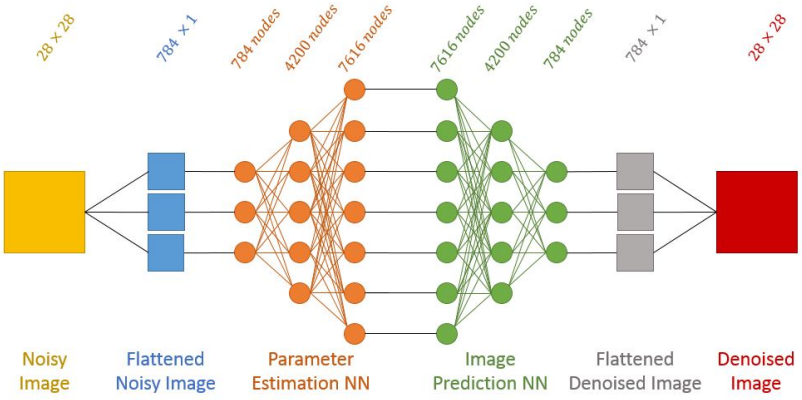


Figure 2: Structure of Neural Network 1

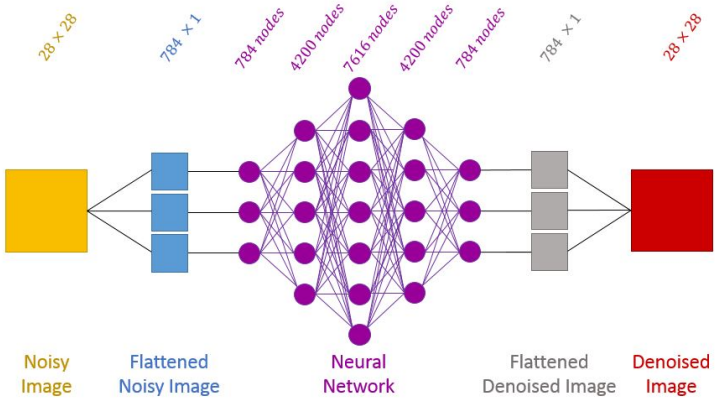


Figure 3: Structure of Neural Network 2

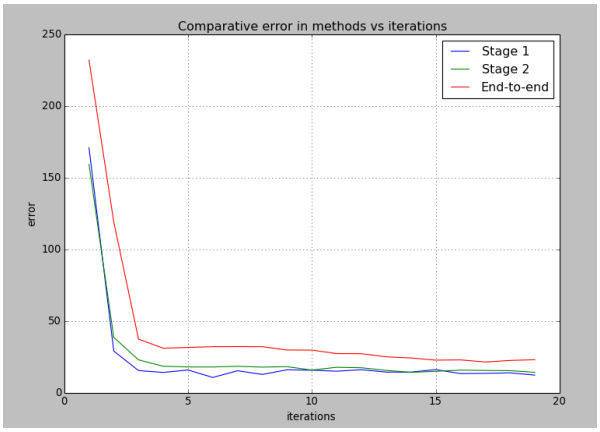


Figure 4: Convergence graph for neural networks. Stage 1 and stage 2 refer to the orange and green networks in Figure 2, respectively. End-to-end refers to the purple neural network trained in Figure 3

5 Related Works

The related works for this project can be categorized into three categories, namely inference in Probabilistic Graphical Models, Deep Learning, and works combining graphical models with Deep Learning.

Inference in PGM: Kim and Pearl [8] were the first to solve probabilistic inference in general graphical models. They proposed local belief updating method based on causal and diagnostic modes. Cooper [2] in his seminal 1990 paper, showed that inference in Bayesian Networks is NP-hard. Zhang and Poole in 1994 [13] and Dechter [3] in 1999 proposed early variants of the widely-known Variable Elimination algorithm.

More recently, Kohli et. al [9] proposed cut based efficient inference algorithm for undirected graphical models. Similarly, Kou et. al [11] proposed efficient inference algorithm for Markov Random Fields based on collective classifications. However, non-of these methods can perform inference in O(1) time given enough training time.

Deep Learning: Deep Learning aims to learn abstract feature representation by applying layers of non-linear transformation on the input. Deep learning has been successful in solving previously seemingly intractable problems in computer vision, speech recognition, natural language processsing, et. cetera. Variants of deep learning methods have been proposed [12, 6, 7] since 1980s. Deep learning are the methods of choice for many tasks in Artificial Intelligence and Machine Learning today. However, the abstract representation (weights) learned during training deep neural networks has a little semantic meaning and are usually not interpretable. This is one of the major drawbacks of using deep learning for naturally relational data.

Combination of Graphical Models and Deep Learning: There has been much research interest lately to harness both the performance of Deep Learning and graphical model's ability to represent dependencies between the variables. Chen et al [1] proposed gradient descent based approximate learning algorithm for structured models which learns structured model jointly with deep features by blending learning and inference steps. Zheng et. al.[14] proposed CRF-RNN which combined Conditional Random Fields with Recurrent Neural Networks by modeling CRF with Gaussian pairwise potentials and mean field inference as recurrent neural network.Domke [4] proposed perturbation based back-propagation algorithm for parameter optimization in graphical models. However, none of these methods use deep networks to perform inference in graphical models, and therefore cannot be directly applied to solve the problem we are interested in.

6 Conclusion

In this project we demonstrated that it is indeed possible to perform inference in a single forward pass of a neural network. To do so, we trained a deep neural network to perform inference in a graphical model given the CPDs for each factor. A key takeaway from this project is that given enough offline training time, it is possible to perform inference as a constant time operation.

We also extended our model to perform MAP inference by training two neural networks: one to learn CPDs and the other to learn marginals. We conducted experiments on image denoising to show the model's applicability to real-world problems.

However, our results have limitations. Currently, our model is only able to handle inference for graphical models with fixed structure, i.e. we train and test on graphical models having equal number of variables and states. We plan to extend our model such that training a single deep neural network is able perform inference on a graphical model with arbitrary structure.

7 Future Works

We plan to address following in future works:

- Extend problem 1 using KL divergence. It will be interesting to see how minimizing KL divergence (both M and I projections) performs against minimizing Hellinger's distance.
- Provide theoretical justification for problem 2.

- Merge problem 2 into a single-step problem by changing the loss function. Problem 2 seems to be redundant in a sense that we train two separate deep networks. We believe it is possible to train a single deep network by reformatting the loss function.

References

[1] L.-C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun. Learning deep structured models. *arXiv preprint arXiv:1407.2538*, 2014.

[2] G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2):393–405, 1990.

[3] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41–85, 1999.

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

[5] E. Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 136:210–271, 1909.

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

[7] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley. *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA, 1984.

[8] J. Kim and J. Pearl. A computational model for causal and diagnostic reasoning in inference systems. 1983.

[9] P. Kohli and P. H. Torr. Dynamic graph cuts for efficient inference in markov random fields. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(12):2079–2088, 2007.

[10] D. Koller and N. Friedman. *Probabilistic graphical models principles and techniques*. MIT press, 2009.

[11] Z. Kou and W. W. Cohen. Stacked graphical models for efficient inference in markov random fields. In *SDM*, pages 533–538. SIAM, 2007.

[12] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.

[13] N. L. Zhang and D. Poole. A simple approach to bayesian network computations. In *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, 1994.

[14] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.