Recent Algorithms Utilizing Neural Networks for Statistical Inference

Maria Kesa

February 15, 2016

Abstract

This report reviews recent algorithms for efficient inference in Bayesian latent variable models. The studied models employ a neural network to represent the posterior of the latent variables given the data, called the recognition model. The recognition model parameters are learned jointly with the generative model parameters by optimizing the variational lower-bound on the data log-likelihood.

1 Introduction

The setting of this report is probabilistic modeling of data. We are interested in Bayesian probabilistic models that employ latent variables–variables that influence the observed data, but are not measured–to capture the patterns in observed data. An example of a latent variable model is Topic Models in text modeling. In Topic Models latent variables are distributions over words. For example, a topic that can be called 'Statistics', will place high probability on words such as 'Gaussian', 'Multivariate', 'p-value'. Each document is described by a distribution over latent topics. The model postulates that the following generative process was used to obtain a document: for each word slot, sample a topic from the topic distribution describing a document and then sample a word from the topic. Once we have specified a probabilistic model such as the topic model, the task is to learn the latent variables and model parameters from data. This is the task of inference.
In this report we look at recent work for utilizing neural networks in probabilistic inference. Artificial neural networks are algorithmic systems that derive inspiration from the learning mechanism in the brain. In essence, they learn non-linear mappings between inputs and outputs by adjusting the connection strengths between the computational units, neurons. This is loosely inspired by the basic neurobiological mechanism that is responsible for learning and adaptation in biological organism, synaptic plasticity. The change in synaptic strength, the strength of connection between neurons, represents the mechanism by which organisms adapt their behavior to the environment. Evolution discovered a very powerful algorithm and artificial neural networks is an attempt to exploit this mechanism in man-made systems. Both airplanes and birds fly– one is an engineered, another a natural system– their mode of operation is very different, though they both accomplish the same thing. In the same way artificial neural networks though inspired by biology differ considerably in the ways that they

invoke learning, but neural networks have achieved better results than humans in certain tasks (citation).
The main question driving the research papers in this report is how to make Bayesian inference more efficient. In latent variable models, we seek to learn the model by calculating the posterior of the latent variables and model parameters given the data. This is a difficult task, because calculating the posterior exactly requires summing over all settings of latent variables and model parameters, an intractable task in any model of reasonable size. There are two principled ways for approximating the posterior. Monte Carlo methods simulate the posterior and draw samples from it. The samples are used as the approximation. Variational inference employs optimization of a bound on data log-likelihood (termed Free Energy) to find an approximation to the posterior. Essentially, it is a search through a family of simpler distributions with their own parameters to find a distribution that best "matches" the posterior.
The papers under consideration in this report employ a novel approach for learning models with latent variables. Although the details of the algorithms differ, the common line of attack is creating an approximation $q_\phi(h|x)$ for the true posterior distribution $p(h|x)$, where h is the latent variable and x is the data.The approximation is a mapping from observed data to the latent values of the model. The papers under consideration employ the strategy of representing this function as a neural network. The weights of this neural network recognition model are learned jointly with the parameters of the model by optimizing a variational bound on the data log-likelihood.

2 Neural Variational Inference and Learning in Belief Networks

The paper under consideration introduces a computationally efficient method for learning Sigmoid Belief Networks. The layered architecture of the network is illustrated in Figure 1. Sigmoid Belief Networks consist of layers of nodes in a directed graph, where the nodes are binary stochastic artificial neurons, connected with edges having a weight. The bottom layer in the network is the data layer, the layers above it are latent variables that model the statistical structure in the data. The conditional distribution of the state of a latent node, given its parent nodes in the layer above, follows a log-linear model. The question that this paper addresses is, given a data set how can we learn the statistical structure in the data by adjusting the weights between the nodes in a Sigmoid Belief Net?

The probability that a neuron s_i is active depends on its parents s_j by the following equation:

$$p(s_i = 1) = \frac{1}{1 + exp(-b_i - \sum s_j w_{ij})}$$

This equation specifies that in order to compute the probability of the activation of a particular neuron we compose a weighted sum of the activations of the parent neurons, add a bias term to the sum and pass the result through a sigmoid non-linearity. The only exception to this are the neurons in the top

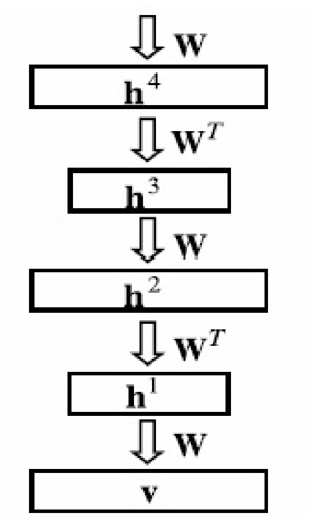

Figure 1: Architecture of the Sigmoid Belief Network

layer, that do not have parents. Their probability of activations is simpler, depending only on their bias.
The Belief Networks are generative models– once they are learned, it is possible to generate observations from them and it can be done very efficiently. This means that if you learn a belief network from credit customer data, containing information such as education and income, you can generate realistic samples that would describe the possible customers that your firm may encounter. This is useful in simulations and artificial intelligence applications, in both scientific and practical contexts.
Given a data set and a network, we have to learn weights and bias terms of the network. We now turn to the inference algorithm that the paper introduces.

2.1 Inference algorithm

Belief Networks are learned by adjusting the weights in such a way so as to make the observed data as likely as possible. To adjust the weights, we need to know the states of the hidden variables above the observed layer. This means that we have to get a sample from a posterior distribution of the hidden variables given the observed variables, $p(h|x)$. There are two problems that make it difficult to sample from the posterior: 1. Observations introduce correlations between nodes in the layer above it that are originally independent making the structure of the posterior more complicated. This phenomenon is called 'Explaining Away'– given an observation, the activation of one unit explains it away, making the activation of the other node less likely, thereby creating an anti-correlation. Because of the introduced dependencies the posterior doesn't factorize according to a simple form, which makes it difficult to sample from, 2. The influence of hidden layers above the level for which we are conducting inference creates a prior that we have to take into account and that is difficult to compute, because it involves summation over an exponential number of configurations of hidden nodes.

The algorithm introduced the paper uses a neural network to obtain exact samples from the posterior of the hidden variables for a particular observation. The weights of this neural network are learned jointly with the model by maximizing a lower bound on the data log-likelihood. The sample from the network is used to estimate gradients for moving in the parameter space of the model, moving towards an optimum that describes the data well.

The neural network that produces samples from the posterior distribution is a mapping from the an observation to the states of the latent variables, $q_\phi(h|x)$, with parameters (neural network weights) ϕ. We denote the parameters of the original model θ (in this model they are the weights in the network and biases), parameterizing a joint distribution between the observed and hidden variables $p_\theta(x,h)$. We can now formulate a loss function, which is to be optimized $L(data,\phi,\theta) = \sum_i L(x_i,\phi,\theta)$. The loss function is derived from the log-likelihood of the data– our goal is to make the data as probable as possible under the model.

The true log-likelihood is intractable, because we have to sum over an exponential number of possible hidden states and parameters to obtain it. A well-known technique that is employed for intractable optimization problems is forming a lower bound on the function to be optimized and then maximizing to make it as tight as possible. Here the bound was derived using Jensen's inequality:

$$\log p_\theta(x) = \log \sum_h p_\theta(x,h) \geq \sum_h q_\phi(h|x) \log \frac{p_\theta(x,h)}{q_\phi(h|x)} = E_q[\log p_\theta(x,h) - \log q_\phi(h|x)] = L(x,\theta,\phi)$$

Now, we have to take the derivatives with respect to theta and phi to perform gradient ascent. The derivates express as follows:

$$\nabla_\theta L(x) = E_q[\nabla_\theta \log p_\theta(x,h)]$$

$$\nabla_\theta L(x) = E_q[(\log p_\theta(x,h) - \log q_\phi(h|x)) \times \nabla_\phi \log q_\phi(h|x)]$$

These expectations are intractable to compute. The authors turn to Monte Carlo integration to compute them. The samples for integration come from the inference network. Given n samples from the inference $q_\phi(h|x)$, we compute the expectations with the following formula:

$$\nabla_\theta L(x) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \log p_\theta(x, h^{(i)})$$

and

$$\nabla_\phi L(x) = \frac{1}{n} \sum_{i=1}^{n} (\log p_\theta(x,h^{(i)}) - \log q_\phi(h^{(i)}|x)) \times \nabla_\phi \log q_\phi(h^{(i)}|x)$$

3 Auto-Encoding Variational Bayes

The variational lower bound of the data log-likelihood is an important concept, because it allows the optimization of an intractable objective. The current

paper introduces an unbiased differentiable estimator of the variational lower bound. This estimator can then be optimized using stochastic gradient ascent techniques.
As in the previous paper, the setting of this paper are models with latent variables h. We assume a generative process for the data in which we first generate a latent variable h from a prior distribution $p_\theta(h)$ and then generate the data point from the conditional distribution $p_\theta(x|h)$. A recognition model $q_\phi(h|x)$ is introduced to approximate the posterior distribution of latent variables given the data. A variational bound $L(\phi, \theta, x)$ is used to find the parameters of the distributions that make the observed data as likely as possible, e.g. learn the model.
The next section describes how the reparameterization of the latent variables leads to an estimator of the variational lower bound.

3.1 Estimator of the Variational Lower Bound

The variational lower bound is derived from data log-likelihood in the following way:

$$\log p_\theta(x) = \log \sum_h p_\theta(x,h) \geq \sum_h q_\phi(h|x) \log \frac{p_\theta(x,h)}{q_\phi(h|x)} = E_q[\log p_\theta(x,h) - \log q_\phi(h|x)] = L(\phi, \theta, x)$$

The goal is to differentiate and optimize the lower bound with respect to both the variational paraemeters ϕ and θ. The gradient with respect to ϕ presents difficulties– the usual Monte Carlo gradient estimator for this type of problem exhibits very high variance. This motivates the following reparameterization trick.

The paper reparameterizes the random variable $h \sim q_\phi(h|x)$ using a differentiable transformation $g(\epsilon, x)$:

$$h = g_\phi(\epsilon, x)$$

with

$$\epsilon \sim p(\epsilon)$$

The function g_ϕ is chosen such that it maps a data point x and a random noise vector ϵ to a sample from an approximate posterior for that point, $h \sim q_\phi(h|x)$. Monte Carlo estimates of the lower bound can now be formed $L^A(\phi, \theta, x) \simeq L(\phi, \theta, x)$:

$$L^A(\phi, \theta, x^{(i)}) = \frac{1}{L} \sum_{i=1}^{n} (\log p_\theta(x, h^{(i,l)}) - \log q_\phi(h^{(i,l)}|x))$$

where $h^{(i,l)} = g_\phi(\epsilon^{(i,l)}, x^{(i)})$ and $\epsilon^{(l)} \sim p(\epsilon)$.

This reparametrization trick can be used when the recognition model $q_\phi(h|x)$ is a neural network, similarly to the previous paper.

4 Stochastic Backpropagation and Approximate Inference in Deep Generative Models

This paper introduces a deep generative model, where the hidden variables are distributed according to Gaussian distributions. The posterior of the latent variables given the data is approximated by a recognition model with its own parameters (similarly to the previous papers). The paper provides an algorithm for jointly learning the parameters of the generative model and the recognition model. The presented model performs well on images, producing realistic samples of handwritten digits, street signs and objects. This indicates that the complex statistics of images can be learned using hierarchical models consisting of Gaussian distributions, proving that this model is flexible and suitable for modeling complex structures in data.

4.1 The model

There are three ingredients in the generative model: Gaussian latent variables $\xi_l \sim N(0, I)$, matrices G_l that transform these latent variables and non-linear multi-layer perceptron transformations T_l. The generative process consists of the following steps: we first draw ξ_l from a multivariate normal distribution, transform it with G_l and pass it on to the next layer, where it is transformed with T_l and perturbed with Gaussian noise to form the activation of layer l,h_l. At the end of the process we draw the observed data x from a distribution, which is parameterized by the transformation at the bottom layer of the hierarchy. This process is described by the following equations:

$$\xi_l \sim N(0, I); l = 1, ..., L$$

$$h_L = G_L \xi_L$$
$$h_l = T_L(h_{l+1}) + G_l \xi_l; l = 1...L - 1$$
$$v \sim \pi(x|T_0(h_1))$$

The parameters of the model that have to be learned from data are the parameters of the transformation T_l and the matrices G_l. The paper refers to these parameters jointly as θ^g. These parameters transform the spherical Gaussian distribution created by ξ to a distribution that matches the empirical distribution of the data.

4.2 Inference Algorithm

As in the previous papers, inference relies on the optimization of the variational lower bound to the data log-likelihood, also called Free Energy. For this model, the variational lower bound has the following form:

$$L(x) = -\log p(x) = -\log \int p(X|\xi, \theta^g) p(\xi, \theta^g) d\xi = -\log \int \frac{q(\xi|x)}{q(\xi|x)} p(X|\xi, \theta^g) p(\xi, \theta^g) d\xi \geq$$
$$F(x) = D_{KL}[q(\xi|x)||p(\xi)] - E_q[\log p(x|\xi, \theta^g) p(\theta^g)] \quad (1)$$

The distribution $q(\xi|x)$ is a recognition model that is instantiated by a deep neural network. It is taken to be a multivariate Gaussian distribution with its own parameters θ^r (r for recognition), including mean μ_l and covariance C_l, which are maps represented by deep neural networks.
To learn the model we need expressions for the gradients of Free Energy (variational lower bound) with respect to the generative model parameters θ^g and recognition model parameters θ^r.
The gradient with respect to the generative model parameters is expressed as:

$$\nabla_{\theta_j^g} F(x) = -E_q[\nabla_{\theta_j^g} p(x|h)] + \frac{1}{\kappa}\theta_j^g$$

The algorithm approximates this gradient by drawing samples from the recognition model q.
The gradients with respect to the parameters of the recognition model are given below. We take the gradients with respect to μ and for computational complexity reasons the factor matrix of the covariance matrix, $C = RR^T$.

$$\nabla_{\mu_l} F(x) = E_q[\nabla_{\xi_l} \log p(x|h(\xi))] + \mu_l$$

$$\nabla_{R_{i,j,l}} F(x) = -\frac{1}{2} E_q[\epsilon_{l,j} \nabla_{\xi_{l,i}} \log p(x|h(\xi))] + \frac{1}{2} \nabla_{R_{i,j,l}} [TrC_{n,l} - \log |C_{n,l}|]$$

Gradients $\nabla_{R_{i,j,l}}[TrC_{n,l} - \log |C_{n,l}|]$ are computed by backpropagation.

5 Conclusion

This report presents a brief overview of a novel and powerful approach to learning latent variable models. In this approach a recognition model representing the posterior of the latent variables $q(h|x)$ is a neural network, whose parameters are optimized jointly with the model parameters. The variational lower bound on the data log-likelihood is employed to learn the parameters of the recognition model and the model parameters jointly. The recognition model can be sampled from efficiently, providing the gradients for learning of the original model. The presented algorithms present an important advance, because they permit to scale up latent variable models to large data sets.

6 References

Mnih, A., Gregor, K. 2014. “Neural Variational Inference and Learning in Belief Networks”, Proceedings of the 31st International Conference on Machine Learning
Rezende, D.J., Mohamed, S., Wierstra, D. 2014. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”, Proceedings of the 31st International Conference on Machine Learning
Kingma, D.P., Welling, M. 2014. “Auto-Encoding Variational Bayes”, The International Conference on Learning Representations