Approximate Inference

Presenter: Ming-Chang Chiu

September 10, 2019

Many probabilistic models are difficult to train because it is difficult to perform inference in them. In deep learning, we usually have a set of visible variables $\boldsymbol{v}$ and a set of latent variables $\boldsymbol{h}$ and the difficuly often refers to computing $p(\boldsymbol{h}|\boldsymbol{v})$.

Most graphical models have many hidden layer and that makes the $p(\boldsymbol{h}|\boldsymbol{v})$ intractable, since exact inference requires an exponential amount of time in these models (See Ch16). In this chapter, we introduce several of the techniques for confronting these intractable inference problems.

1 Inference as Optimization

One approach to address the difficulty in inference is to cast the exact inference as an optimization problem, so that approximating the exact inference problem becomes approximating the underlying optimization problem. E.g. Assume we have a probabilistic model $p(\boldsymbol{v}, \boldsymbol{h})$, and we want to know $\log$ probability of observed data, $\log p(\boldsymbol{v}; \boldsymbol{\theta})$, but sometimes it is hard to marginalize out $\boldsymbol{h}$. Instead of computing it directly, we compute the lower bound on $L(\boldsymbol{v}, \boldsymbol{\theta}, q)$ on $\log p(\boldsymbol{v}; \boldsymbol{\theta})$, which is called *evidence lower bound* (ELBO), defined to be

$$L(\boldsymbol{v}, \boldsymbol{\theta}, q) = \log p(\boldsymbol{v}; \boldsymbol{\theta}) - D_{KL}(q(\boldsymbol{h}|\boldsymbol{v})||p(\boldsymbol{h}|\boldsymbol{v}; \boldsymbol{\theta})) \tag{1}$$

where q is an arbitrary probability distribution over $\boldsymbol{h}$.

Since KL divergence is always non-negative, we can see that L is always at most the same value as the resired probability, and so, to find the approximate value of $\log p(\boldsymbol{v}; \boldsymbol{\theta})$ becomes an optmization with objective to maximize L.
For an appropriate choice of q, L is tractable to compute. And algebraically, we can rearrange L into a more convenient form (see appendix):

$$L(\boldsymbol{v}, \boldsymbol{\theta}, q) = -\mathbf{E}_{\boldsymbol{h}\sim q}[\log q(\boldsymbol{h}|\boldsymbol{v}) - \log p(\boldsymbol{h}, \boldsymbol{v}; \theta)] \tag{2}$$

We can now think of inference as procedure for finding the q that maximizes L. We can make the optimization procedure less expensive but approximate by restricting the family of distributions q the optimization is allowed to search over or by using an imperfect optimization procedure that may not completely maximize L but merely increase it by a significant amount.

2 Expectation Maximization

The first algorithm to maximize the lower bound L is *expectation maximization* (EM) algorithm. Note that EM is not an approach to approximate inference, but rather an approach to learning with an approximate posterior.
The EM algorithm consists of alternating between two steps until convergence:

- E-step: Let $\theta^{(0)}$ denote the value of the parameters at the beginning of the step. Set $q(\boldsymbol{h}^{(i)}|\boldsymbol{v}) = p(\boldsymbol{h}^{(i)}|\boldsymbol{v}^{(i)}; \theta^{(0)})$ for all indices i of the training examples $\boldsymbol{v}^{(i)}$ we want to train on (both batch and minibatch variants are valid). By this we mean q is defined in terms of the current parameter value of $\theta^{(0)}$; if we vary θ then $p(\boldsymbol{h}|\boldsymbol{v}; \theta)$ will change but $q(\boldsymbol{h}|\boldsymbol{v})$ will remain equal to $p(\boldsymbol{h}|\boldsymbol{v}; \theta^{(0)})$.

- M-step: Completely or partially maximize

$$\sum_i L(\boldsymbol{v}^i, \theta, q) \tag{3}$$

  with respect to using your optimization θ algorithm of choice.

This can be viewed as a coordinate ascent algorithm to maximize L. On one step, we maximize L with respect to q, and on the other, we maximize L with respect to θ.
One can use stochastic gradient ascent on latent variable models, which is seen as a special case of EM where M-step consists of taking a single gradient step. Other variants of the EM algorithm can make much larger steps.

Even though the E-step involves exact inference, we can think of the EM algorithm as using approximate inference in some sense. Specifically, the M-step assumes that the same value of q can be used for all values of θ. This will introduce a gap between L and the true $\log p(\boldsymbol{v})$ as the M-step moves further and further away from the value $\theta^{(0)}$ used in the E-step. Fortunately, the E-step reduces the gap to zero again as we enter the loop for the next time. (give emample)

We can get an insight that there is the basic structure of the learning process, in which we update the model parameters to improve the likelihood of a completed dataset, where all missing variables have their values provided by an estimate of the posterior distribution.

3 MAP Inference and Sparse Coding

As before, we are interested in comuting $p(\boldsymbol{h}|\boldsymbol{v})$. An alternative form of inference is to compute the **single** most likely value of the missing variables, rather than to infer the entire distribution over their possible values. In the context of latent variable models, this means computing

$$\boldsymbol{h}^* = \underset{\boldsymbol{h}}{\operatorname{argmax}}\, p(\boldsymbol{h}|\boldsymbol{v}) \tag{4}$$

This is known as *maximum a posteriori* inference, abbreviated MAP inference.

Recall from Eq.2,

$$L(\boldsymbol{v}, \boldsymbol{\theta}, q) = -\mathbf{E}_{\boldsymbol{h}\sim q}[\log q(\boldsymbol{h}|\boldsymbol{v}) - \log p(\boldsymbol{h}, \boldsymbol{v}; \theta)] \tag{5}$$

We can derive MAP inference as a form of approximate inference by restricting the family of distributions q may be drawn from. Specifically, we require q to take on a Dirac distribution:

$$q(\boldsymbol{h}|\boldsymbol{v}) = \delta(\boldsymbol{h} - \boldsymbol{\mu}) \tag{6}$$

This means that we can now control q entirely via $\boldsymbol{\mu}$. Dropping terms of L that do not vary with $\boldsymbol{\mu}$, we are left with the optimization problem

$$\boldsymbol{\mu}^* = \underset{\boldsymbol{\mu}}{\operatorname{argmax}} \log p(\boldsymbol{h} = \boldsymbol{\mu}, \boldsymbol{v}) \tag{7}$$

which is equivalent to Eq.4.

As with EM, this is a form of coordinate ascent on L, where we alternate between using inference to optimize L with respect to q and using parameter updates to optimize L with respect to θ. MAP is primarily used for sparse coding models.

Sparse coding is a linear factor model that imposes a sparsity-inducing prior on its hidden units. A common choice is a factorial Laplace prior, with

$$p(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|} \tag{8}$$

The visible units are then generated by performing a linear transformation and adding noise:

$$p(\boldsymbol{v}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{v}; \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}, \beta^{-1}\boldsymbol{I}) \tag{9}$$

Computing or even representing $p(\boldsymbol{h}|\boldsymbol{v})$ is difficult. Every pair of variables h_i and h_j are both parents of $\boldsymbol{v}$.

Because $p(\boldsymbol{h}|\boldsymbol{v})$ is intractable, we use MAP inference and learn the parameters by maximizing the ELBO defined by the Dirac distribution around the MAP estimate of $\boldsymbol{h}$.

If we concatenate all of the $\boldsymbol{h}$ vectors in the training set into a matrix $\boldsymbol{H}$, and concatenate all of the $\boldsymbol{v}$ vectors into a matrix $\boldsymbol{V}$, then the sparse coding learning process consists of minimizing

$$J(\boldsymbol{H}, \boldsymbol{W}) = \sum_{i,j} |H_{i,j}| + \sum_{i,j} (\boldsymbol{V} - \boldsymbol{H}\boldsymbol{W}^T)_{i,j}^2 \tag{10}$$

We can minimize J by alternating between minimization with respect to $\boldsymbol{H}$ and minimization with respect to $\boldsymbol{W}$. Both sub-problems are convex.

4 Variational Inference and Learning

The core idea behind variational learning is that we can maximize L over a restricted family of distributions q. This family should be chosen so that it is easy to compute $\mathbf{E}_q \log p(\boldsymbol{h}, \boldsymbol{v})$. A typical way to do this is to introduce assumptions about how q factorizes.
A common approach to variational learning is to impose the restriction that q is a factorial distribution:

$$q(\boldsymbol{h}|\boldsymbol{v}) = \prod_i q(h_i|\boldsymbol{v}) \tag{11}$$

This is called the mean field approach. Generally speaking, we can impose any graphical model structure we choose on q, to flexibly determine how many interactions we want our approximation to capture. We do not need to specify a specific parametric form for q. We specify how it should factorize, but then the optimization problem determines the optimal probability distribution within those factorization constraints. For discrete latent variables, we use traditional optimization techniques; for continuous latent variables, we use calculus of variations to performe optimization, and actually determine which function should be used to represent q. Calculus of variations makes designer of the model must only specify how q factorizes, rather than q.

Recall from Eq.2, we can think of maximizing L w.r.t. q as minimizing $D_{KL}(q(\boldsymbol{h}|\boldsymbol{v})||p(\boldsymbol{h}|\boldsymbol{v}))$.

4.1 Discrete Latent Variables

Variational inference with discrete latent variables is relatively straightforward. We define a distribution q, typically one where each factor of q is just defined by a lookup table over discrete states. After determining how to represent q , we simply optimize its parameters. In principle this can be done with any optimization algorithm, such as gradient descent.
Because this optimization must occur in the inner loop of a learning algorithm, it must be very fast. To achieve this speed, we typically use special optimization algorithms that are designed to solve comparatively small and simple problems in very few iterations. A popular choice is to iterate fixed point equations, in other words, to solve

$$\frac{\partial}{\partial \hat{h}_i} L = 0 \tag{12}$$

for $\hat{h}_i$. We repeatedly update different elements of $\hat{\boldsymbol{h}}$ until we satisfy a convergence criterion.

4.2 Calculus of Variations

Calculus of Variations enables us to actually solve for a function f(x), such as when we want to find the probability density function over some random variable. A function of a function f is known as a functional $J[f]$. We can take functional derivatives, also known as variational derivatives, of a functional $J[f]$ with respect to individual values of the function $f(\boldsymbol{x})$ at any specific value of $\boldsymbol{x}$. The functional derivative of the functional J with respect to the value of the function f at point

$\boldsymbol{x}$ is denoted $\frac{\delta}{\delta f(\boldsymbol{x})} J$. For instance, for differentiable functions $f(\boldsymbol{x})$ and differentiable functions $g(y, \boldsymbol{x})$ with continuous derivatives, that

$$\frac{\delta}{\delta f(\boldsymbol{x})} \int g(f(\boldsymbol{x}), \boldsymbol{x}) d\boldsymbol{x} = \frac{\partial}{\partial y} g(f(\boldsymbol{x}), \boldsymbol{x}) \tag{13}$$

Similar to optimizing a function w.r.t. a vector, we can optimize a functional by solving for the function where the functional derivative at every point is equal to zero.

Example: consider the problem of finding the probability distribution function over $x \in \mathbb{R}$ that has maximal differential entropy. Recall that

$$H[p] = -\mathbf{E}_x \log p(x) = -\int p(x) \log p(x) dx \tag{14}$$

Adding contraints using Lagrange multipliers regarding mean, variance, and probability distribution.

$$L[p] = \lambda_1 (\int p(x) dx - 1) + \lambda_2 (\mathbf{E}[x] - \mu) + \lambda_3 (\mathbf{E}[(x-\mu)^2] - \sigma^2) + H[p] \tag{15}$$

$$= \int (\lambda_1 p(x) + \lambda_2 p(x) x + \lambda_3 p(x)(x-\mu)^2 - p(x) \log p(x)) dx - \lambda_1 - \mu\lambda_2 - \sigma^2 \lambda_3 \tag{16}$$

And set functional derivatives equal to 0:

$$\forall x, \frac{\delta}{\delta p(x)} L = \lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2 - 1 - \log p(x) = 0 - \sigma^2 \lambda_3 \tag{17}$$

And so,

$$p(x) = exp(\lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2 - 1) \tag{18}$$

To finish this optimization problem, we are left with choosing the λ values, as long as the constraints are satisfied. In the book, the authors set $\lambda_1 = 1 - \log \sigma\sqrt{2\pi}, \lambda_2 = 0, \lambda_3 = -\frac{1}{2\sigma^2}$, to obtain $p(x) = \mathcal{N}(x; \mu, \sigma^2)$. This is one reason for using the normal distribution when we do not know the true distribution.

4.3 Continuous Latent Variables

When our graphical model contains continuous latent variables, we must now use calculus of variations when maximizing L with respect to $q(\boldsymbol{h}|\boldsymbol{v})$. There is a general equation for the mean field fixed point updates, so people need not solve calculus of variations themselves. If we make the mean field approximation

$$q(\boldsymbol{h}|\boldsymbol{v}) = \prod_i q(h_i|\boldsymbol{v}) \tag{19}$$

and fix $q(h_j|\boldsymbol{v})$ for all $j \neq i$, then the optimal $q(h_i|\boldsymbol{v})$ may be obtained by

$$\tilde{q}(\boldsymbol{h}|\boldsymbol{v}) = exp(\mathbf{E}_{\boldsymbol{h}_{-i} \sim q(\boldsymbol{h}_{-i}|\boldsymbol{v})} \log \tilde{p}(\boldsymbol{v}, \boldsymbol{h})) \tag{20}$$

so long as $p(\boldsymbol{v}, \boldsymbol{h}) \neq 0$ at everywhere. Now Eq.20 is a fixed point equation, and can be iteratively for all i until convergence. We can also regard some of the values appear in it as parameters and optimize with any algorithm we like.
Example: Suppose that $p(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; 0, \boldsymbol{I})$ and $p(v|\boldsymbol{h}) = \mathcal{N}(v; \boldsymbol{w}^T\boldsymbol{h}; 1)$, with $\boldsymbol{h} \in R^2$, and single v. The true posterior is given by

$$p(\boldsymbol{h}|\boldsymbol{v}) \tag{21}$$

$$\propto p(\boldsymbol{h}, \boldsymbol{v}) \tag{22}$$

$$= p(h_1)p(h_2)p(\boldsymbol{v}|\boldsymbol{h}) \tag{23}$$

$$\propto exp(-\frac{1}{2}[h_1^2 + h_2^2 + (v - h_1w_1 - v - h_2w_2)^2]) \tag{24}$$

Applying Eq.20, we find that

$$\tilde{q}(h_1|\boldsymbol{v}) = exp(\mathbf{E}_{h_2 \sim q(h_2|\boldsymbol{v})} \log \tilde{p}(\boldsymbol{v}, \boldsymbol{h})) \tag{25}$$

$$= exp(-\frac{1}{2}\mathbf{E}_{h_2 \sim q(h_2|\boldsymbol{v})}[h_1^2 + h_2^2 + (v - h_1w_1 - v - h_2w_2)^2]) \tag{26}$$

$$= exp(-\frac{1}{2}\mathbf{E}_{h_2 \sim q(h_2|\boldsymbol{v})}[h_1^2 + h_2^2 + v^2 + h_1^2w_1^2 + h_2^2w_2^2 - 2vh_1w_1 - 2vh_2w_2 + 2h_1w_1h_2w_2]) \tag{27}$$

$$= exp(-\frac{1}{2}[h_1^2 + \langle h_2^2 \rangle + v^2 + h_1^2w_1^2 + \langle h_2^2 \rangle w_2^2 - 2vh_1w_1 - 2v\langle h_2 \rangle w_2 + 2h_1w_1\langle h_2 \rangle w_2]) \tag{28}$$

From this we can see that $\tilde{q}$ has the functino form of gaussian, which is not assumed at the outset. Using the same approach on a different model could yield a different functional form of q.

4.4 Interactions between Learning and Inference

Using approximate inference as part of a learning algorithm affects the learning process, and this in turn affects the accuracy of the inference algorithm.
Specifically, the training algorithm tends to adapt the model in a way that makes the approximating assumptions underlying the approximate inference algorithm become more true. When training the parameters, variational learning increases

$$E_{\boldsymbol{h} \sim q} \log p(\boldsymbol{v}, \boldsymbol{h}) \tag{29}$$

For a specific $\boldsymbol{v}$, this increases $p(\boldsymbol{h}|\boldsymbol{v})$ for values of $\boldsymbol{h}$ that have high probability under $q(\boldsymbol{h}|\boldsymbol{v})$ and decreases $p(\boldsymbol{h}|\boldsymbol{v})$ for values of $\boldsymbol{h}$ that have low probability under $q(\boldsymbol{h}|\boldsymbol{v})$.

This behavior causes our approximating assumptions to become self-fulfilling prophecies. If we train the model with a unimodal approximate posterior, we will obtain a model with a true posterior that is far closer to unimodal than we would have obtained by training the model with exact inference.

5 Learned Approximate Inference

We can think of the optimization process as a function f that maps an input $\boldsymbol{v}$ to an approximate distribution $q^* = \text{argmax}_q L(\boldsymbol{v}, q)$. Other methods of approximate inference can be very expensive, such as fixed point equations or gradient-based methods. To avoid such, we can think of the multi-step iterative optimization process as just being a function, and approximate it with a neural network that implements an approximation $\hat{f}(\boldsymbol{v}; \boldsymbol{\theta})$.

5.1 Wake-Sleep

It is sometimes hard to train a model to infer $\boldsymbol{h}$ from $\boldsymbol{v}$, since we do not have a supervised training set. The wake-sleep algorithm resolves this problem by drawing samples of both $\boldsymbol{h}$ and $\boldsymbol{v}$ from the model distribution.
Example: in a directed model, this can be done cheaply by performing ancestral sampling beginning at $\boldsymbol{h}$ and ending at $\boldsymbol{v}$. The inference network can then be trained to perform the reverse mapping: predicting which $\boldsymbol{h}$ caused the present $\boldsymbol{v}$.

5.2 Other Forms of Learned Inference

Other than wake-sleep, Salakhutdinov and Larochelle (2010) showed that a single pass in a learned inference network could yield faster inference than iterating the mean field fixed point equations in a DBM.
Variational Autoencoder, one of the recent approach to generative modeling, is also a form of learned approximate inference. In this elegant approach, there is no need to construct explicit targets for the inference network. Instead, the inference network is simply used to define L elegant approach.