Lecture #9: Probability and Variational Inference Primer

Weixin Wang, Joo Gek Lim

1 Introduction

In this lecture, we mainly study latent variable models, variational inference, amortized variational inference and generative models. The goal is to understand latent variable models and how to use variational inference.

2 Probabilistic Latent Model



Figure 1: The description of Latent variable models.

Figure 1 shows an example of Gaussian mixture model where z is the latent.

the model:
$$p_{\theta}(x)$$

the data: $\mathcal{D} = \{x_1, x_2, x_3, \dots, x_N\}$
maximum likelihood fit:
 $\theta \leftarrow \arg \max_{\theta} \frac{1}{N} \sum_i \log p_{\theta}(x_i)$

$$p(x) = \int p(x|z)p(z)dz$$

$$\theta \leftarrow \arg \max_{\theta} \frac{1}{N} \sum_{i} \log \left(\int p_{\theta}(x_i | z) p(z) dz \right)$$

completely intractable

Figure 2: The description of Latent variable models.

Exploration in Computer Science Research: Deep Reinforcement Learning (CS 6101, 2019), National University of Singapore.

Variational Inference 3

However, it is quite tough to train such latent variable models as described in Figure 2. Alternatively, we use approximate learning to learn the model. Here we first introduce KL-divergence, which is used to measure the difference between two distributions, as follows:

$$D_{\mathrm{KL}}(q\|p) = E_{x \sim q(x)} \left[\log \frac{q(x)}{p(x)} \right] = E_{x \sim q(x)} [\log q(x)] - E_{x \sim q(x)} [\log p(x)] = -E_{x \sim q(x)} [\log p(x)] - \mathcal{H}(q)$$

The variational inference is as follows: Our objective is to minimize KL-divergence.

$$\begin{aligned} \mathcal{L}_{i}(p,q_{i}) \\ \log p(x_{i}) &\geq E_{z \sim q_{i}(z)}[\log p(x_{i}|z) + \log p(z)] + \mathcal{H}(q_{i}) \\ \log p(x_{i}) &= D_{\mathrm{KL}}(q_{i}(x_{i}) || p(z|x_{i})) + \mathcal{L}_{i}(p,q_{i}) \\ \log p(x_{i}) &\geq \mathcal{L}_{i}(p,q_{i}) \\ D_{\mathrm{KL}}(q_{i}(x_{i}) || p(z|x_{i})) &= E_{z \sim q_{i}(z)} \left[\log \frac{q_{i}(z)}{p(z|x_{i})} \right] = E_{z \sim q_{i}(z)} \left[\log \frac{q_{i}(z)p(x_{i})}{p(x_{i},z)} \right] \\ &= -E_{z \sim q_{i}(z)}[\log p(x_{i}|z) + \log p(z)] - \mathcal{H}(q_{i}) + \log p(x_{i}) \\ &-\mathcal{L}_{i}(p,q_{i}) \end{aligned}$$
 independent of q_{i} !

 \Rightarrow maximizing $\mathcal{L}_i(p, q_i)$ w.r.t. q_i minimizes KL-divergence!

Figure 3: The variational inference.



calculate $\nabla_{\theta} \mathcal{L}_i(p, q_i)$: sample $z \sim q_i(x_i)$ $\nabla_{\theta} \mathcal{L}_i(p, q_i) \approx \nabla_{\theta} \log p_{\theta}(x_i | z)$ $\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathcal{L}_i(p, q_i)$ update q_i to maximize $\mathcal{L}_i(p, q_i)$

How many parameters are there?

let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$ use gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$ gradient ascent on μ_i, σ_i

 $|\theta| + (|\mu_i| + |\sigma_i|) \times N$

intuition: $q_i(z)$ should approximate $p(z|x_i)$ what if we learn a network $q_i(z) = q(z|x_i) \approx p(z|x_i)$?



Figure 4: The problem with variational inference.

Amortized Variational Inference However, there will be a practical problem with variational inference as discussed in Figure 4. Amortized variational inference is to, for example, a neural network that accepts the observation as input, and outputs the mean and variance parameter for the latent variable instead of optimizing a set of free parameters. We then optimize the parameters of the neural network.

for each x_i (or mini-batch):

$$\begin{array}{c} \text{calculate } \nabla_{\theta} \mathcal{L}(p_{\theta}(x_{i}|z), q_{\phi}(z|x_{i})): \\ \text{sample } z \sim q_{\phi}(z|x_{i}) \\ \nabla_{\theta} \mathcal{L} \approx \nabla_{\theta} \log p_{\theta}(x_{i}|z) \\ \theta \leftarrow \theta + \alpha \nabla_{\theta} \mathcal{L} \\ \hline \phi \leftarrow \phi + \alpha \nabla_{\phi} \mathcal{L} \end{array} \begin{array}{c} \text{look up formula for entropy of a} \\ \text{Gaussian} \\ \mathcal{L}_{i} = E_{z \sim q_{\phi}(z|x_{i})}[\log p_{\theta}(x_{i}|z) + \log p(z)] + \mathcal{H}(q_{\phi}(z|x_{i})) \\ \hline \mathcal{L}_{i} = E_{z \sim q_{\phi}(z|x_{i})}[\log p_{\theta}(x_{i}|z) + \log p(z)] + \mathcal{H}(q_{\phi}(z|x_{i})) \\ \hline \mathcal{L}_{i} = E_{z \sim q_{\phi}(z|x_{i})}[r(x_{i},z)] \\ \hline \mathcal{L}_{i} = E_{z \sim q_{\phi}(z|x_{i})}[r(x_{i},z)] \end{array}$$

can just use policy gradient!

What's wrong with this gradient?

Limitation of PG: high variance

 $\label{eq:grad} \operatorname{grad}\,J(\phi)\approx \frac{1}{M}\sum_{j}\nabla_{\phi}\log q_{\phi}(z_{j}|x_{i})r(x_{i},z_{j})$



Is there a better way?

$$\begin{split} J(\phi) &= E_{z \sim q_{\phi}(z|x_i)}[r(x_i, z)] & q_{\phi}(z|x) = \mathcal{N}(\mu_{\phi}(x), \sigma_{\phi}(x)) \\ &= E_{\epsilon \sim \mathcal{N}(0,1)}[r(x_i, \mu_{\phi}(x_i) + \epsilon \sigma_{\phi}(x_i))] & z = \mu_{\phi}(x) + \epsilon \sigma_{\phi}(x) \\ \text{estimating } \nabla_{\phi} J(\phi) : & & \downarrow \\ &\text{sample } \epsilon_1, \dots, \epsilon_M \text{ from } \mathcal{N}(0, 1) & (\text{a single sample works well!}) & \epsilon \sim \mathcal{N}(0, 1) \end{split}$$

$$\nabla_{\phi} J(\phi) \approx \frac{1}{M} \sum_{j} \nabla_{\phi} r(x_i, \mu_{\phi}(x_i) + \epsilon_j \sigma_{\phi}(x_i)) \text{ Low variance :)} \qquad \text{independent of } \phi!$$
Because now we are directly computing grad instead of sampling
most autodiff coff ware (o.g., TopcorFlow) will compute this for

most autodiff software (e.g., TensorFlow) will compute this for you!

Figure 6: The reparameterization trick.

Reparameterization Trick To solve the high variance problem as discussed in Figure 5, we introduce the reparameterization trick. Compared with policy gradient, the reparameterization is very simple to implement and has low variance. However, it only works for continuous latent variables. While the policy gradient can handle both discrete and continuous variables. However, policy gradient has high variance and requires multiple samples and small learning rates.

4 Variational Autoencoder



Figure 7: The illustration of Varitional Autoencoder.

A variational autoencoder comprises an encoder, a decoder, and a loss function. As shown in Figure 7, the encoder $q_{\phi}(z|x)$ takes as input x, and encodes x into a latent representation z. While the decoder $p_{\theta}(x|z)$ takes as input the latent representation z and predicts x. The training objective consists of two parts where the first part is the expected log-likelihood and the second part is KL-divergence for measuring the information loss.