# 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

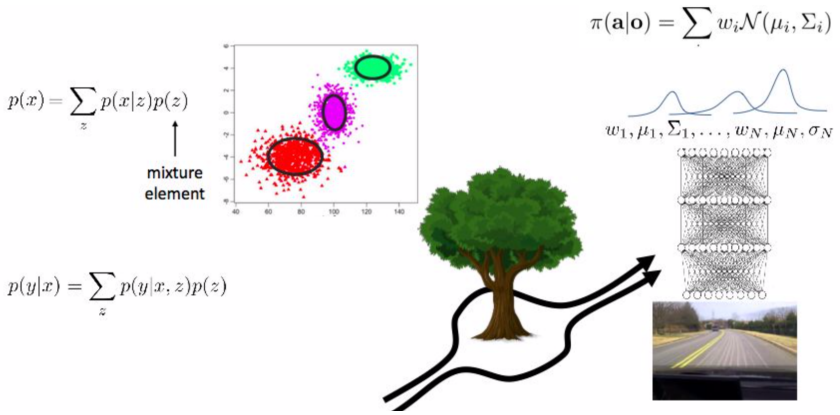



Figure 1: The description of Latent variable models.

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

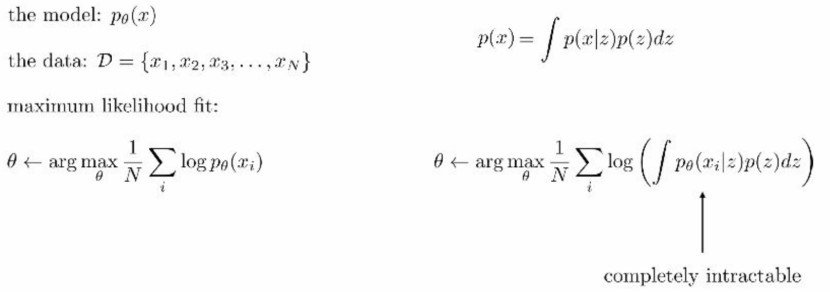



Figure 2: The description of Latent variable models.

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

## 3 Variational Inference

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.

$$\log p(x_i) \geq \overbrace{E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)}^{\mathcal{L}_i(p,q_i)}$$

$$\log p(x_i) = D_{\mathrm{KL}}(q_i(x_i)\|p(z|x_i)) + \mathcal{L}_i(p,q_i)$$
$$\log p(x_i) \geq \mathcal{L}_i(p,q_i)$$

$$D_{\mathrm{KL}}(q_i(x_i)\|p(z|x_i)) = E_{z\sim q_i(z)}\left[\log \frac{q_i(z)}{p(z|x_i)}\right] = E_{z\sim q_i(z)}\left[\log \frac{q_i(z)p(x_i)}{p(x_i,z)}\right]$$
$$= \underbrace{-E_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] - \mathcal{H}(q_i)}_{-\mathcal{L}_i(p,q_i)} + \log p(x_i) \quad \text{independent of } q_i!$$

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

Figure 3: The variational inference.

for each $x_i$ (or mini-batch):
- 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 $\mathcal{L}_i(p,q_i)$

let's say $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

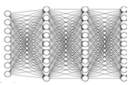
use gradient $\nabla_{\mu_i}\mathcal{L}_i(p,q_i)$ and $\nabla_{\sigma_i}\mathcal{L}_i(p,q_i)$

gradient ascent on $\mu_i$, $\sigma_i$

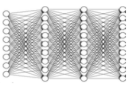
How many parameters are there? $|\theta| + (|\mu_i| + |\sigma_i|) \times N$

intuition: $q_i(z)$ should approximate $p(z|x_i)$

what if we learn a *network* $q_i(z) = q(z|x_i) \approx p(z|x_i)$?




$p_\theta(x|z)$




$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$

Figure 4: The problem with variational inference.

**Amortized Variational Inference** However, there will be a practical problem with varaiational 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.

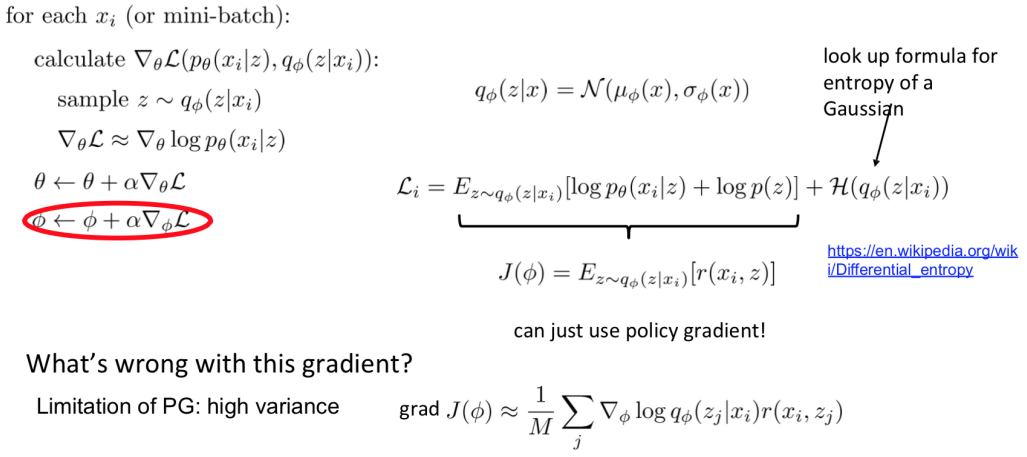



Figure 5: Amartized varitional inference.

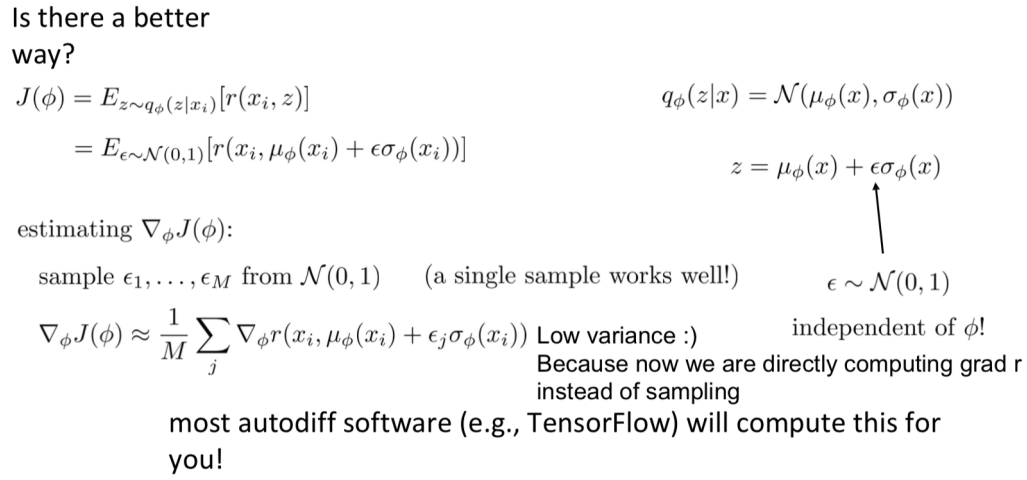



Figure 6: The reparameterization trick.

**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

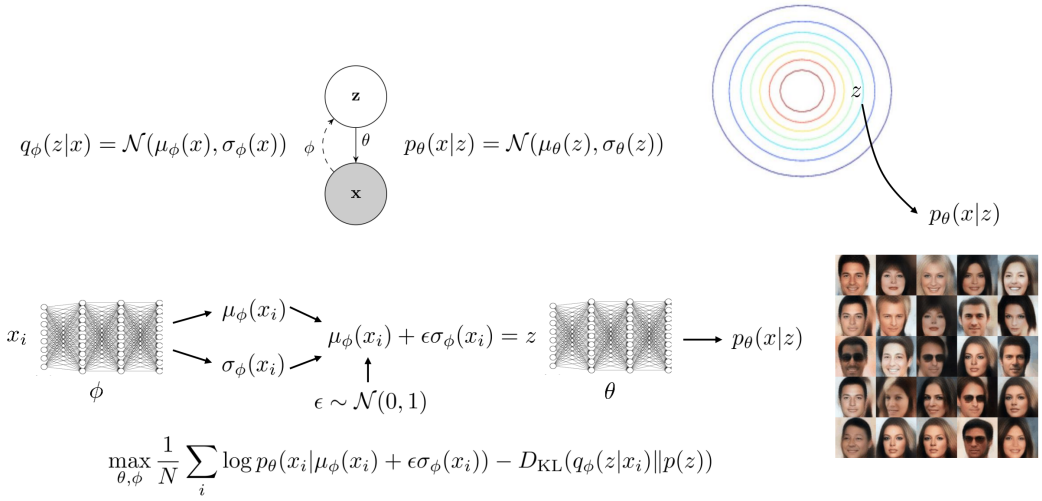



Figure 7: The illustration of Varitional Autoencoder.

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.