CS 329M: Topics in Artificial Intelligence **Spring 2017**

Lecture 6: May 8

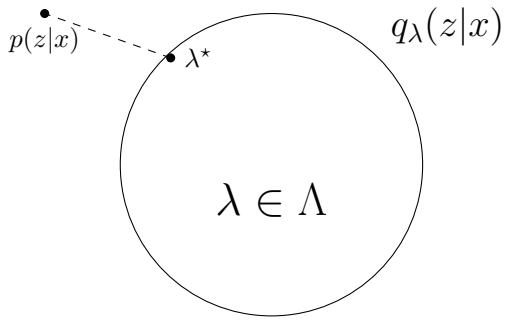
Lecturer: James Zou *Scribe: Nikhil Garg, Arjun Seshadri, Pranav Sriram*

6.1 Announcements

- Project proposal presentations next Monday (5/15) in class
  - Prepare a 5 minute presentation, either chalk board or slides
  - Send instructors slides before-hand if applicable
  - Provide a precise definition the problem you are solving, along with a high level description of datasets and your approach
  - More details to be posted on Canvas

6.2 Recap: Variational Inference

In variational inference, we are interested in finding the conditional distribution $p(z|x)$, where x denotes observed data and z denotes latent variables. We assume that we can evaluate $p(z|x)$ up to a proportional constant α. The approach we took last lecture was to consider a family of distributions $q_\lambda(z|x)$ parameterized by λ, and find $\lambda^\star \in \Lambda$ that minimizes the Kullback-Leibler divergence (KL distance) between $p(z|x)$ and $q_{\lambda^\star}(z|x)$.



Specifically, we have

$$\begin{aligned} &\min_\lambda \; KL(q(z|x)||p(z|x)) \\ &= \min_\lambda \; \mathbb{E}_{q_\lambda} \left[\log q_\lambda - \log p\right] \end{aligned}$$ [1]

where we want q such that it is easy to sample $z \sim q_\lambda(z|x)$, and easy to compute the gradient $\nabla_\lambda q_\lambda(z|x)$. With this recap, we make a few observations:

[1] Since we can only measure p up to a constant, we can replace p with αp_m, and rewrite the objective as $\min_\lambda \mathbb{E}_{q_\lambda} [\log q_\lambda - \log p_m - \log \alpha]$. Now, we can simply pull out $\log \alpha$ from the expectation and remove it from the objective as it does not depend on λ. For the remainder of these notes, $\log p$ will be used to denote $\log p_m$

- q_λ is just a function that takes x as input, and outputs a probability distribution over z, parameterized by λ. Specifically, q_λ : Input $x \to$ dist/func on z.
- Furthermore, we have a 3rd lesson in ML in addition to the first two:
  1. Use gradient descent when possible
  2. Preferably, use stochastic gradient descent
  3. When you have a parametrized function mapping, use neural networks (today's lecture)

Today: **Variational Inference with Neural Networks**

6.3 Neural Net Approach

We have a set of observations $x = \{x^{(1)} \dots x^{(N)}\}$. We want $q_\lambda(z|x^{(i)})$ to approximate $p(z|x^{(i)})$. An example would be topic modeling, where we are given a document (really, a bag of words) $x^{(i)}$, and we want the distributions over the possible topics z.

Take $x \in \mathbb{R}^{d_1}, z \in \mathbb{R}^{d_2}$, where typically $d_2 \ll d_1$ and define the mapping:

$$q_\lambda(z|x) = \mathcal{N}(z; \mu_\lambda(x), \sigma^2_\lambda(x))$$

That is, $q_\lambda(z|x)$ is a normal distribution over z with mean $\mu_\lambda(x)$ and variance $\sigma^2_\lambda(x)$. The trick is to use neural networks to represent μ and σ^2.

$$\mu_\lambda : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}, \sigma^2_\lambda : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2} \qquad \text{feed forward NNs (separate)}$$
$$\lambda : \{\text{weights for both networks}\}$$

Note that we assume the covariance matrix is diagonal, which allows us to write $\sigma^2_\lambda(x)$ as a vector. There are advantages of forcing $q_\lambda(z|x^{(i)})$ to be Gaussian, as opposed to directly modeling it as an arbitrary distribution using a Neural Network with a softmax layer. Most notably, all the complexity is in finding the mean and variance, which can be black-boxed.

Question: Does this particular parameterization satisfy the two properties we wanted? Yes!

To sample $z \sim q_\lambda(z|x)$ do the following: take $\epsilon \sim N(0, I_{d_2})$, and set $z = \mu_\lambda(x) + \sigma_\lambda(x) \circ \epsilon$, where $\circ$ is the pointwise product.

To calculate gradient $\nabla_\lambda q_\lambda(z|x)$ we can simply use backpropagation.

More precisely on the gradient $\nabla_\lambda KL$ for each observation $x^{(i)}$:

$$\text{For each } i, \text{ sample } l \text{ } z\text{'s} : z^{(i,l)} = \mu_\lambda(x^{(i)}) + \sigma_\lambda(x^{(i)}) \cdot \epsilon^{(l)}, \text{where } \epsilon^{(l)} \sim N(0, I_{d_2})$$
$$\nabla_\lambda KL \simeq \frac{1}{S} \sum_l \nabla_\lambda \log(q_\lambda(z^{(i,l)}|x^{(i)})) \times \Big[\log(q_\lambda(z^{(i,l)}|x^{(i)})) - \log(p_\lambda(z^{(i,l)}|x^{(i)}))\Big]$$

The above gives the gradient for one observation. Now, sum over $x^{(i)}$.

Further note that because q is a Gaussian, its log probability has the following form:

$$\nabla_\lambda \log q_\lambda = \nabla_\lambda \left(-\frac{\|z^{(i)} - \mu(x^{(i)})\|}{\sigma^2(x^{(i)})} \right)$$

6.4 Connection to Auto-encoders

First let's review standard autoencoders, which are models that essentially try to learn the identity function. Why is this useful? If the model first maps a high dimensional input to a lower dimensional space, then the model will have to reconstruct the high dimensional input based on a lower dimensional latent space representation. This forces the model to learn some useful structure in the data.

In a single layer autoencoder, there is an encoding step, $z = f(Wx)$, where W is a matrix of weights and f is some non-linearity.

The reconstruction is $\hat{x} = g(Vz)$, and the objective is

$$\min_{W,V} \sum_i \|x^{(i)} - \hat{x}^{(i)}\|^2$$

Note that if f, g are the identity functions, then the autoencoder is just PCA. With general f, g, we are trying to learn non-linear representations. More generally, we can layer several affine and nonlinear transformations so that the mappings from x to z and from z to x take the form of deep neural networks. For simplicity, we will restrict the current discussion to single layer transformations, though most of what follows can be easily extended to the general setting.

Now recall from the previous section that we have a generative model $p : z \to x$. We also have an inference network q: $x \to z$. Putting these things together, we have something resembling an auto-encoder $x \to^q z \to \hat{x}$ which takes an input x, tries to create a latent representation z, which is used to reconstruct x. The difference here from standard autoencoders is that the mappings are stochastic. The input x defines a probability distribution over latent codes z as opposed to a single latent code, and once we sample from this distribution, the sampled code defines a distribution over reconstructions $\hat{x}$.

Let's make the connection to KL more explicit:

$$\begin{aligned} KL(q||p) &= \mathbb{E}\left[\log q(z|x) - \log p(z) - \log p(x|z)\right] + c && [c \text{ constant}] \\ &= KL(q(z|x)||p(z)) - \mathbb{E}\left[\log p(x|z)\right] \end{aligned}$$

In the variational inference framework, we sample $z^{(i)} \sim q(z|x)$ and use Monte-Carlo approximation for the expectation, i.e. $\mathbb{E}\left[\log p(x|z)\right] \approx \log p(x^{(i)}|z^{(i)})$.

If we consider the generative model: $p(z) \sim N(0, I), p(x|z) \sim N(x; \mu_g(z), \sigma^2)$, then we are analogous to the decoder step. The idea is that if the generative model is correct and the variational approximation is from above, then these objectives match: $\log p(x^{(i)}|z^{(i)}) = -\|x^{(i)} - \hat{x}^{(i)}\| + c$, where $\hat{x}^{(i)} = \mu_g(z^{(i)}) = g(V \cdot z^{(i)})$.

In this manner, we have a probabilistic version of the traditional auto-encoder, with an additional regularization term. This model is known as a *variational autoencoder*.