3. Minimizing KL, both ways round

D.J. Wischik

Consider two KL-minimization problems, where q is a fixed distribution and p_θ is a model we're seeking to fit:

$$\min_\theta \mathrm{KL}(q \,\|\, p_\theta) \quad \text{and} \quad \min_\theta \mathrm{KL}(p_\theta \,\|\, q).$$

The first form is used for fitting. The second is called *variational inference*.

1 The fitting problem

Let q be the unknown true distribution, and let p_θ be our proposed model. The KL is

$$\mathrm{KL}(q \,\|\, p_\theta) = \mathbb{E}_{X\sim q} \log \frac{q(X)}{p_\theta(X)}$$

and when we minimize it via gradient descent we need the gradient

$$\nabla_\theta \,\mathrm{KL}(q \,\|\, p_\theta) = -\nabla_\theta \mathbb{E}_{X\sim q} \log p_\theta(X) = -\mathbb{E}_{X\sim q} \nabla_\theta \log p_\theta(X).$$

(To see why the expectation and the gradient can swap, simply write out the expectation as a sum or integral weighted by $q(x)$.) This gradient can easily be estimated by Monte Carlo, as long as we are able to compute $\nabla_\theta \log p_\theta(x)$ per datapoint x. Concretely, for a sample $x_1, \ldots, x_n$ drawn from q,

$$\nabla_\theta \,\mathrm{KL}(q \,\|\, p_\theta) \approx -\frac{1}{n} \sum_{i=1}^n \nabla_\theta \log p_\theta(x_i).$$

Why do we seek $\mathrm{KL}(q \,\|\, p_\theta)$ in the first place, when we're fitting a model to data? First, the only access we have to the unknown true distribution q is our dataset $x_1, \ldots, x_n$, so we had better use an objective that only needs samples from q. Second, this form uses the asymmetry of KL in a sensible way. There's useful notation for describing this: write $f \ll g$ to mean "$f(x) = 0$ whenever $g(x) = 0$". Then, when $q \ll p_\theta$ the KL can be finite, whereas when $q \not\ll p_\theta$ it is necessarily infinite. In words, at any point x where there is the possibility of real data ($q(x) > 0$), then our $\mathrm{KL}(q \,\|\, p_\theta)$ objective insists that p_θ *must* be able to generate it. It seems sensible to insist that our model must be able to describe the dataset we have in front of us. On the other hand it's reasonable to permit p_θ to generate data that q doesn't; our objective treats this as a weakness in the model rather than as a fundamental flaw: it penalizes a model p_θ that puts probability mass on regions that are never reached by q, but it doesn't rule it out completely.

2 The variational problem

The *variational inference* form is typically used when q is some known reference distribution, and we want to find an approximation p_θ to it. In this case, the asymmetry of KL means that our objective insists on $p_\theta \ll q$, in other words "I absolutely do not want my model p_θ to be able to generate anything that q doesn't allow, but I don't mind so much if my model discards some outputs that q might generate."

When minimizing $\mathrm{KL}(p_\theta \,\|\, q)$, this form requires us to "differentiate the sampling operation". Algebraically,

$$\begin{aligned}
\nabla_\theta \left[\mathbb{E}_{X\sim p_\theta} \log \frac{p_\theta(X)}{q(X)} \right] &= \nabla_\theta \sum_x p_\theta(x) \log \frac{p_\theta(x)}{q(x)} \\
&= \sum_x \nabla_\theta \left[p_\theta(x) \log \frac{p_\theta(x)}{q(x)} \right] \\
&\neq \sum_x p_\theta(x) \,\nabla_\theta \left[\log \frac{p_\theta(x)}{q(x)} \right] = \mathbb{E}_{X\sim p_\theta} \nabla_\theta \log \frac{p_\theta(X)}{q(X)}.
\end{aligned}$$

In words, we *can't* simply take a Monte Carlo estimate of the gradient using a fixed sample from p_θ. We have to take account of how changing θ will change the sample that we're averaging over.

Here are two techniques that let us differentiate the sampling operation. Both of them apply generically to $\nabla_\theta \mathbb{E}_{X\sim p_\theta} h_\theta(X)$ with an arbitrary differentiable function h_θ, so that's the form we'll describe; simply plug in $h_\theta(X) = \log p_\theta(X)/q(X)$ for the KL estimate. Remember, this form is generally used when we know q, and we know p_θ by design, and so h_θ is a known function.

Reparameterization trick. When p_θ has a nice form, there is an elegant workaround.[1] Suppose that we can write it as $p_\theta = \Pr_{f_\theta(E)}$ for some noise random variable E and some differentiable function f_θ, where the noise distribution doesn't depend on θ. In other words, suppose $X \sim p_\theta$ can be generated as $X = f_\theta(E)$. A concrete example: $X \sim N(\mu_\theta, \sigma_\theta^2)$ can be generated as $X \sim \mu_\theta + \sigma_\theta E$ where $E \sim N(0,1)$. Then,

$$\mathbb{E}_{X\sim p_\theta} h_\theta(X) = \mathbb{E}_E\, h_\theta\big(f_\theta(E)\big)\,.$$

The distribution of E is fixed so the gradient and the expectation can swap:

$$\nabla_\theta \mathbb{E}_{X\sim p_\theta} h_\theta(X) = \mathbb{E}_E \nabla_\theta\, h_\theta\big(f_\theta(E)\big).$$

Applying this to KL, and using a sample $\varepsilon_1, \ldots, \varepsilon_n$ drawn from E,

$$\nabla_\theta \operatorname{KL}(p_\theta \,\|\, q) \approx \frac{1}{n}\sum_{i=1}^n \nabla_\theta \log \frac{p_\theta\big(f_\theta(\varepsilon_i)\big)}{q\big(f_\theta(\varepsilon_i)\big)}.$$

The trick only works when $X \sim p_\theta$ can be written as $X = f_\theta(E)$ for some differentiable f_θ and some fixed noise distribution E. It works for Gaussians; it's not suitable when for example X has a categorical distribution over tokens.

Reinforcement learning method. In the case of general p_θ, the earlier algebra shows us how to differentiate properly:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{X\sim p_\theta} h_\theta(X) &= \sum_x \nabla_\theta \Big[p_\theta(x) h_\theta(x)\Big] \quad \text{as above} \\
&= \sum_x p_\theta(x)\Big[\nabla_\theta h_\theta(x)\Big] + \sum_x \Big[\nabla_\theta p_\theta(x)\Big] h_\theta(x) \\
&= \sum_x p_\theta(x)\Big[\nabla_\theta h_\theta(x)\Big] + \sum_x \Big[p_\theta(x)\nabla_\theta \log p_\theta(x)\Big] h_\theta(x) \quad \text{since } p_\theta \nabla_\theta \log p_\theta = \nabla_\theta p_\theta \\
&= \mathbb{E}_{X\sim p_\theta}\Big[\nabla_\theta h_\theta(X)\Big] + \mathbb{E}_{X\sim p_\theta}\Big[h_\theta(X)\nabla_\theta \log p_\theta(X)\Big].
\end{aligned}$$

The first term is natural; the second is the correction for the change in sampling distribution. This final expression is now a straightforward expectation, which we can estimate using Monte Carlo sampling: take a sample $x_1, \ldots, x_n$ from p_θ and then

$$\nabla_\theta \mathbb{E}_{X\sim p_\theta} h_\theta(X) \approx \frac{1}{n}\sum_{i=1}^n \Big\{\nabla_\theta h_\theta(x_i) + h_\theta(x_i)\nabla_\theta \log p_\theta(x_i)\Big\}.$$

In the KL case, where $h_\theta(x) = \log p_\theta(x)/q(x)$, there is a further simplification: the first term comes out to zero. This is because

$$\mathbb{E}_{X\sim p_\theta}\Big[\nabla_\theta \log \frac{p_\theta(X)}{q(X)}\Big] = \sum_x p_\theta(x)\frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} = \nabla_\theta \sum_x p_\theta(x) = \nabla_\theta 1 = 0.$$

[1] This trick was popularized by Kingma and Welling, *Auto-encoding variational Bayes*, 2014, in the context of training a variational autoencoder. In this context, they need the gradient of the ELBO, and part of this gradient is

$$\nabla_{\theta,\phi} \mathbb{E}_{X\sim q_X, Z\sim r^\phi_{Z|X}} \log p^\theta_X(X|Z)$$

where q_X is fixed but $r^\phi_{Z|X}$ depends on the parameters of interest. They reparameterize it as $Z = \mu_\phi(x) + \rho_\phi(x)E$ for $E \sim N(0, I)$, so that the desired gradient is

$$\mathbb{E}_{X\sim q_X, E} \nabla_{\theta,\phi} \log p^\theta_X(X|Z = \mu_\phi(x) + \rho_\phi(x)E).$$

This can then be estimated using a single (x, ε) sample.

Interpretation as reinforcement learning. In reinforcement learning, we are typically interested in choosing a policy p_θ to maximize expected reward $\mathbb{E}_{X\sim p_\theta} h(X)$, for some reward function h that doesn't depend on θ. In this setting,

$$\nabla_\theta \mathbb{E}_{X\sim p_\theta} h(X) = \mathbb{E}_{X\sim p_\theta}\Big[h(X)\nabla_\theta \log p_\theta(X)\Big].$$

To interpret this, think of how the change in θ affects the probability of each outcome: if we run a gradient-ascent update step using a single Monte Carlo sample x_0,

$$\theta \leftarrow \theta + \delta h(x_0)\nabla_\theta \log p_\theta(x_0)$$

then the change in $\log p_\theta(x)$ for any outcome x is

$$\log p_\theta(x) \leftarrow \log p_\theta(x) + \delta h(x_0)\nabla_\theta \log p_\theta(x) \cdot \nabla_\theta \log p_\theta(x_0) + O(\delta^2).$$

This update is known as REINFORCE.[2] Sample x_0 from our policy p_θ, score the sample by its reward $h(x_0)$, and push the policy by changing $\log p_\theta(x_0)$ in proportion to this score (with collateral impact on the policy at other x). In the simple case where p_θ has enough degrees of freedom to approximate any distribution, repeated updating should eventually find the policy that puts most of its mass on $\arg\max_x h(x)$, i.e. the policy collapses.[3]

In the fitting problem, our goal is to maximize an expected score $\log p_\theta(X)$, using samples from a fixed distribution q. In reinforcement learning likewise the goal is to maximize an expected score — but now it's the policy, i.e. the distribution that we take our samples from, that we're interested in improving. This sort of policy-improvement optimization shows up both in SFT (as presented in lectures) and in RLHF.

[2] R.J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", *Machine Learning* 8 (1992).

[3] In the LLM setting, suppose we've learned a reward function $r(x, y)$ that scores the response y to a given prompt x. If we naively seek a language model that maximizes reward, $\max_\theta \mathbb{E}_X \mathbb{E}_{Y\sim p_\theta(\cdot|X)} r(X, Y)$, and if our language model has enough capacity, this maximization will find a collapsed model: it will return the single highest-reward completion for each prompt. This is undesirable (it's likely there are pathologies in the reward function we've learnt, so such a collapsed model will come across as 'gaming the system') and it's why RLHF includes a regularization term.