Machine learning for non-differentiable loss functions

from Khanh's little things

For a long time, we were training machine learning models by optimizing surrogate 替代函数 functions or log-likelihood rather than evaluation functions. For example, while the evaluation is the 0-1 loss, the objective function maybe log loss, hinge loss, or any other functions that upper-bound the 0-1 loss. This is obviously undesirable but why were we doing that? Is there an alternative way?

To begin with, let' s mathematically formulate our training objective: 这个公式表示的是什么意思？
为什么给loss function加了期望？？
直接拿RL的优化目标入手？

$$\min E_{y \sim P_\theta}[f(y)]$$

where:

- θ is the model parameters. Pθ is the prediction distribution of the model.
- y is the prediction.
- f is the loss function.

There is a small problem here: Pθ **also depends on the input** x. However, we can resolve this by letting θ 把...纳入 subsume x, i.e. x is a component of the vector θ.

To optimize this objective, we need to compute its gradient with respect to model parameters, $\nabla_\theta E_{y \sim P_\theta}[f(y)]$, in order to apply some gradient-based optimizer (e.g. SGD, Adagrad, Adam). In the past, we were using this gradient estimator:

$$\nabla_\theta E_{y \sim P_\theta}[f(y)] = E_{y \sim P_\theta}[\nabla_\theta f(y)]$$

This equation seems to do nothing except pushing the gradient notation inside the expectation. However, it tells us that to calculate the gradient of an expectation, we can calculate the gradients of individual samples and sum them up.

这种方法的瓶颈是它需要f是所有θ的连续函数。但是像0-1 loss这种阶梯函数不满足这种属性。

The bottleneck of this approach is that it requires f to be a **continuous function with respect to**θ. However, step functions such as the 0-1 loss do not satisfy this property. Its gradient at y=0 is undefined.

在y=0的时候，阶梯函数的梯度是未定义的。

Recently, machine learning researchers realize that they can use a different gradient estimator from the above. This is inspired from reinforcement learning, particularly REINFORCE algorithm (Williams, 1992) and policy gradient (Sutton et al., 1999), which is a generalized version of the former. This estimator is derived as follows:

$$\nabla_\theta E_{y\sim P_\theta}[f(y)] = \nabla_\theta \int P_\theta(y) f(y) = \int P_\theta(y) f(y) \frac{\nabla_\theta P_\theta(y)}{P_\theta(y)} = E_{y\sim P_\theta}\left[f(y)\frac{\nabla_\theta P_\theta(y)}{P_\theta(y)}\right]$$

$$= E_{y\sim P_\theta}\left[f(y)\ln \nabla_\theta P_\theta(y)\right]$$

$$(\ln f(x))' = \frac{f(x)'}{f(x)}$$

This is often called the likelihood ratio/score function estimator (M. C. Fu, 2006).

上面的等式是amazing的，因为它不需要求f对θ的导数。

The above equation is amazing because it **does not require taking derivative of** f **with respect to** θ. One might ask: so now we can train models with 0-1, problem solved?

Not yet! Using the above gradient estimator comes with a cost. How come? The derivation contains all equalities. Yes, but in optimization, deriving the gradient is only part of the story. At every optimization

step, we take a step in the direction (or opposite to the direction) of the gradient, but the **step size** (how wide the step is) is also important. If you multiply the gradient by a constant, its direction does not alter. But then if you don't adjust the step size properly, you will end up oscillating, i.e. you will zig-zag around the optimal point without hitting it. Moreover, in practice, the expectation is only approximated by samples from P_θ, which makes the gradient even more unstable. This is known as the variance problem. It is shown that the score function estimator's variance scales up with the dimension of y. In structured prediction, where y is a sequence, this means that the variance scales up with the length of sequence. Bad news!

震荡的 曲折的 期望是通过从pθ中采样来近似得到的 ->梯度不稳定

Fortunately, there are techniques to reduce variance of the estimator. The simplest one is to normalize $f(y)$ batch-wise. More advanced techniques include actor-critic algorithms, or natural gradients.

Practical guide:

It turns out to be very easy to apply the score function gradient estimator, since most of machine learning models optimize log-likehood objective:

$$E_y[f(y)] = E_y\left[\ln P_\theta(y)\right]$$

whose derivative is:

$$\nabla_\theta E_y[f(y)] = E_y\left[\nabla_\theta \ln P_\theta(y)\right]$$

We see that the quantity inside the expectation differs from that in the score function gradient estimator only by a factor of $f(y)$. Hence, to convert the log-likelihood objective to using any other objective

f(y), we simply need to **multiply the original objective' s gradients by**f(y).

Futher reading:

Schulman, John, et al. "Gradient estimation using stochastic computation graphs." *Advances in Neural Information Processing Systems*. 2015.

https://arxiv.org/pdf/1506.05254.pdf