Gradient-based Optimization for Discrete Distributions

CSC412 Winter 2019
Xuechen Li

Slides based on Chris Maddison's Fields Institute talk
http://www.stats.ox.ac.uk/~cmaddis/pubs/relaxed_grad_estimators_talk.pdf

Overview

- Motivation: why is optimization of discrete distributions hard?
- **REINFORCE** (Williams, 1992)
    - a.k.a log-derivative trick/policy gradient algorithm
- **Control variates** for variance reduction
- **Reparameterization trick** (Kingma & Welling; Jimenez et al., 2014)
    - generalizing reparameterization w/ rejection sampling (RSVI, Naesseth et al., 2017)
- **Concrete random variables** (Maddison et al., 2016)
    - a.k.a **Gumbel-softmax random variables (**Jang et al., 2016**)**
- **Generalizing control variates in modern ML**
    - using concrete random variables (**REBAR,** Tucker et al., 2017)
    - using neural networks (**RELAX,** Grathwohl et al., 2017)
- Other Very Recent Developments

Motivation

Motivation: why is optimization of discrete distributions hard?

- Problems we care about are sometimes of the form:

θ: Parameters optim. w.r.t.

$$\mathbb{E}_{p(b;\theta)}\left[f(b,\theta)\right]$$

b: Random variable

f: Loss function

- Goal: optimize this **expected loss** w.r.t parameters θ

Computing gradients is hard

- For gradient-based optim., we need the gradient of expected loss w.r.t. θ:

$$\nabla_\theta \mathbb{E}_{p(b;\theta)}\left[f(b,\theta)\right]$$

- Difficulties:
  - the expected loss might NOT have closed form formulae
  - f might NOT be differentiable
  - b might be a discrete random variable; this makes Monte Carlo solutions hard

REINFORCE

REINFORCE: simple Monte Carlo gradients

- a.k.a **score-function/log-derivative trick** in statistics
- a.k.a **policy gradient algorithm** in RL

$$
\begin{aligned}
&\nabla_\theta \mathbb{E}_{p(b;\theta)}\left[f(b)\right] \\
&= \nabla_\theta \int f(b) p(b;\theta) db \\
&= \int f(b) p(b;\theta) \nabla_\theta \log p(b;\theta) db \\
&= \mathbb{E}_{p(b;\theta)}\left[f(b) \nabla_\theta \log p(b;\theta)\right] \\
&\approx \frac{1}{M} \sum_{i=1}^{M} f(b^{(i)}) \nabla_\theta \log p(b^{(i)};\theta)
\end{aligned}
$$

Note: if f(.) is also a function of θ, there would be an extra term to this gradient. Try deriving yourself.

b^{(i)}:i-th Monte Carlo sample from the distribution

REINFORCE: simple Monte Carlo gradients

- Example (RL):

$$\mathbb{E}_{p(b;\theta)}\left[f(b,\theta)\right]$$

- b is the **action** sampled from a **policy** p(.; θ); in RL it's usually denoted as π(.)
- f(.) is the **reward** function; in RL it's usually denoted as R(.)
- Implementation of "f(b) grad-log p(b)" simple:
    - use \`*tf.stop_gradient*\` in TensorFlow
    - use \`*torch.no_grad*\` context manager in PyTorch

```python
def diffable_loss_tf(policy, f):
    b = policy.sample()
    surr_loss = tf.stop_gradient(f(b)) * policy.logp(b)
    return surr_loss
```

```python
def diffable_loss_torch(policy, f):
    b = policy.sample()
    with torch.no_grad():
        blocked_f = f(b)
    surr_loss = blocked_f * policy.logp(b)
    return surr_loss
```

REINFORCE: learning hard attention

- Using REINFORCE to learn hard attention in Neural Turing Machines:

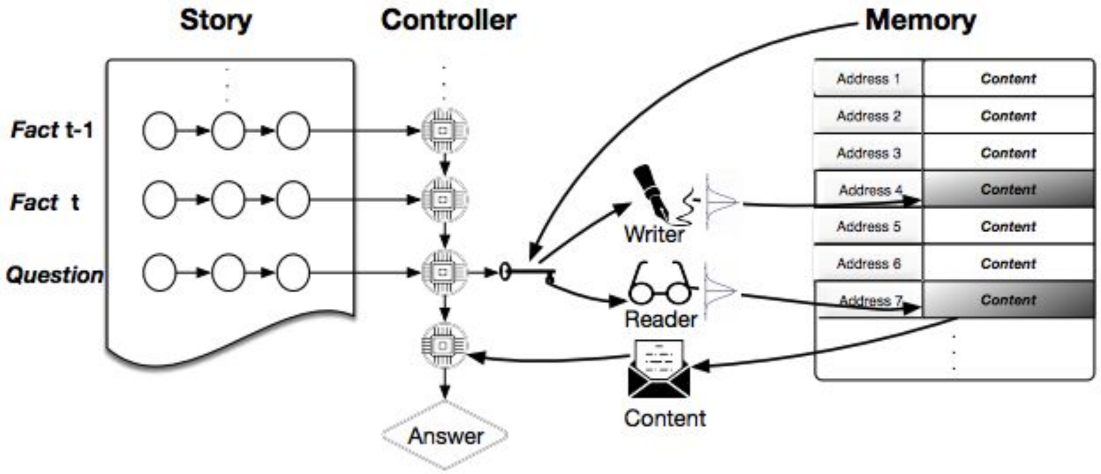

Figure from *Dynamic Neural Turing Machine with Continuous and Discrete Addressing Schemes,* Gulcehre et al., 2017

REINFORCE: learning hard attention

4.1 Training discrete D-NTM

To train discrete D-NTM, we use REINFORCE (Williams, 1992) together with the three variance reduction techniques–global baseline, input-dependent baseline and variance normalization– suggested in (Mnih and Gregor, 2014).

Let us define $R(\mathbf{x}) = \log p(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_T; \boldsymbol{\theta})$ as a reward. We first center and rescale the reward by,

$$\tilde{R}(\mathbf{x}) = \frac{R(\mathbf{x}) - b}{\sqrt{\sigma^2 + \epsilon}},$$

where b and σ is running average and standard deviation of R. We can further center it for each input $\mathbf{x}$ separately, i.e.,

$$\bar{R}(\mathbf{x}) = \tilde{R}(\mathbf{x}) - b(\mathbf{x}),$$

where $b(\mathbf{x})$ is computed by a baseline network which takes as input $\mathbf{x}$ and predicts its estimated reward. The baseline network is trained to minimize the Huber loss (Huber, 1964) between the true reward $\tilde{R}(\mathbf{x})$ and the predicted reward $b(\mathbf{x})$. This is also called as input based baseline (IBB) which is introduced in (Mnih and Gregor, 2014).

Excerpt from *Dynamic Neural Turing Machine with Continuous and Discrete Addressing Schemes,* Gulcehre et al., 2017

Control Variate

Control Variate: variance reduction for Monte Carlo

- Observation: expectation doesn't change if we add and subtract same R.V.

$$\mathbb{E}[\underbrace{\hat{X}}_{\text{old estimator}}] = \mathbb{E}[\underbrace{\hat{X} + Y}_{\text{new estimator}} - Y]$$

Control Variate: variance reduction for Monte Carlo

- If X hat and Y are correlated, then the overall variance may be reduced
- We typically design Y so that its expectation can be computed in closed form

$$\mathbb{E}[\hat{X}] = \mathbb{E}[\hat{X} + Y - Y]$$
$$= \mathbb{E}[\hat{X} + Y] - \mathbb{E}[Y]$$
$$\approx \frac{1}{M}\sum_{i=1}^{M}(\hat{X}^{(i)} + Y^{(i)}) - \mathbb{E}[Y]$$

- What's the variance of the new estimator in Blue?

Control Variate: variance reduction for Monte Carlo

- Variance analysis in the case of a *single* Monte Carlo sample (M=1)
- In RL, a **constant baseline** (or running avg) is used as Y

$$
\begin{aligned}
&\mathrm{Var}(\hat{X} + Y - \mathbb{E}[Y]) \\
&= \mathrm{Var}(\hat{X} + Y) \\
&= \mathrm{Var}(\hat{X}) + \mathrm{Var}(Y) + 2\mathrm{Cov}(\hat{X}, Y)
\end{aligned}
$$

- When is this variance lower than before?

Control Variate: antithetic variates

- Say we want to estimate the expectation of a function of some Gaussian R.V.
- We can sample in an i.i.d. manner:

$$\mathbb{E}_{X\sim\mathcal{N}(\mu,\Sigma)}[f(X)] \approx \frac{1}{M}\sum_{i=1}^{M} f(x^{(i)})$$

$$\text{where} \quad x^{(i)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu,\Sigma)$$

- But this may given **high** variance...

Control Variate: antithetic variates

- We may sample in opposite directions when the distribution is **symmetric**:

$$
\begin{aligned}
&\mathbb{E}_{X\sim\mathcal{N}(0,I)}[f(X)] \\
&= \mathbb{E}_{X\sim\mathcal{N}(0,I)}[\tfrac{f(X)+f(-X)}{2}] \\
&\approx \frac{1}{2M}\left(\sum_{i=1}^{M} f(x^{(i)}) + \sum_{i=1}^{M} f(-x^{(i)})\right)
\end{aligned}
$$

- This may reduce the variance of the estimator. When?

Reparameterization

Reparameterization trick (Kingma et al.; Rezende et al., 2014)

- Works for a restricted set of continuous distributions
- Take Gaussian random variables for instance:
  - Instead of sampling direction from a Gaussian w/ mean **μ** and covariance **Σ**
  - we first sample **ε** from a unit Gaussian, and then perform the linear transformation:
  - **X = ε√Σ + μ**
  - This is also because most Autodiffs do not recognize functions such as `*torch.randn*`!

$$
\begin{aligned}
&\nabla_\theta \mathbb{E}_{p(b;\theta)}[f(b)] \\
&= \nabla_\theta \mathbb{E}_{p(\epsilon)}[f(g(\epsilon, \theta))] \\
&= \mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g(\epsilon, \theta))] \\
&\approx \frac{1}{M} \sum_{i=1}^{M} \nabla_\theta f(g(\epsilon^{(i)}, \theta))
\end{aligned}
$$

RSVI: generalizing reparam. for more distributions

- Up to now, this only works for Gaussians
- But what is we want to “differentiably” sample from Dirichlet, gamma, inverse gamma, Von-mises distributions?
- Rejection Sampling VI (RSVI, Naesseth et al., 2016)

Algorithm 1 Reparameterized Rejection Sampling

Input: target $q(z;\theta)$, proposal $r(z;\theta)$, and constant M_θ, with $q(z;\theta) \leq M_\theta r(z;\theta)$

Output: ε such that $h(\varepsilon,\theta) \sim q(z;\theta)$

1: $i \leftarrow 0$
2: **repeat**
3: $\quad i \leftarrow i+1$
4: $\quad$ Propose $\varepsilon_i \sim s(\varepsilon)$
5: $\quad$ Simulate $u_i \sim \mathcal{U}[0,1]$
6: **until** $u_i < \frac{q(h(\varepsilon_i,\theta);\theta)}{M_\theta r(h(\varepsilon_i,\theta);\theta)}$
7: **return** ε_i

RSVI: generalizing reparam. for more distributions

```python
def sample_gamma(alpha, B, batch_size):
    # returns the sampled gammas and the log(pi) of the samples
    # under the acceptance distribution
    # sample epsilon for each gamma for alpha params
    epsilon = sample_pi(alpha + B, 1., (batch_size,))
    z_tilde = h(epsilon, alpha + B, 1.)
    # gamma samples
    z_gamma = shape_augmentation(z_tilde, B, alpha)
    # get per-sample log-likelihoods
    lp_i = log_pi(epsilon, alpha + B, 1., z=z_tilde)
    return z_gamma, lp_i
```

Code snippet courtesy of Will Grathwohl

Discrete Relaxations

Concrete random variables

- **Con**tinuous relaxation of dis**crete** variable
  - There is no simple way to reparameterize discrete RVs
  - But we can for a **continuously relaxed** one
- The Gumbel-Max trick (Luce 1959)

log-prob for i-th category

independent Gumbel sample

$$Y_i = \log a_i + G_i$$
$$X = \arg\max Y$$

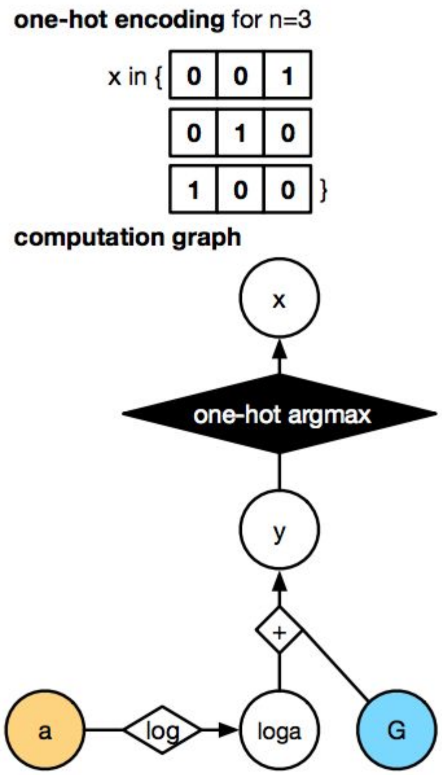

Figure from Chris Maddison’s field institute talk

Concrete random variables

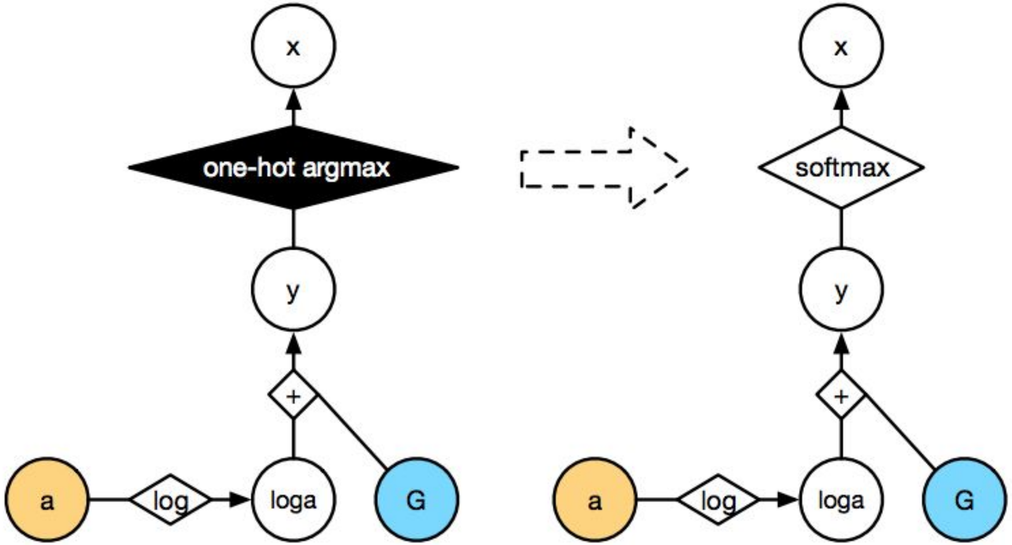

Figure from Chris Maddison's field institute talk

Concrete random variables

- Study this random variable called concrete (Maddison et al., 2017) or Gumbel-softmax (Jang et al., 2017).

$$Y_i = \log a_i + G_i$$

$$\tilde{g}(y, \lambda)_i = \frac{\exp(y_i/\lambda)}{\sum_j \exp(y_j/\lambda)}$$

$$\tilde{X} \stackrel{d}{=} \tilde{g}(Y, \lambda)$$

- Assume that the loss is well-defined on ~X
- But gradient is **biased** w.r.t. the original loss due to softmax approximation!

Concrete random variables: biasedness

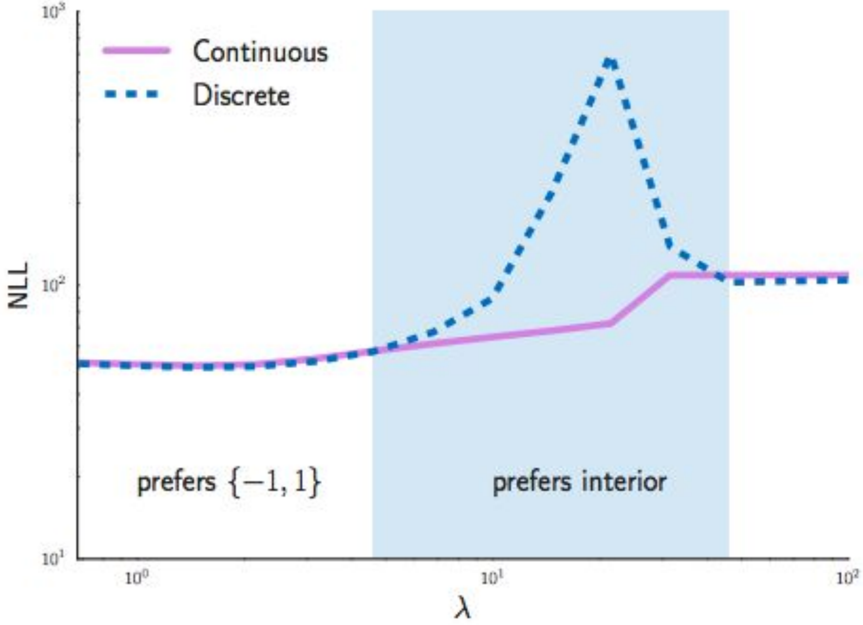

Figure from Chris Maddison's field institute talk

REBAR: generalizing control variates w/ concrete

- REBAR (Tucker et al., 2017) instead uses the Concrete RV as control variate

z: Gumbel + log-prob.

$$\underbrace{\left(f(g(z)) - f(\tilde{g}(z))\right)\nabla_\theta \log p(b;\theta)}_{\text{REINFORCE}} + \underbrace{\nabla_\theta f(\tilde{g}(z))}_{\text{Reparam.}}$$

g: Argmax

tilde{g}: Softmax

REBAR: generalizing control variates w/ concrete

- How is the variance of the estimator controlled?
- We can optimize the variance w.r.t. temperature parameter!

$$\frac{d}{d\lambda}\mathbb{V}(\hat{\theta}) = \mathbb{E}\left[2\hat{\theta}\,\frac{d}{d\lambda}\hat{\theta}\right]$$

Derivative w.r.t. temperature

REBAR Gradient estimator

RELAX: generalizing control variates w/ neural nets

- Why assume the control variate is based on concrete?
- Base control variate on neural net, and optimize variance!

$$\left(f(g(z)) - c(z)\right) \nabla_\theta \log p(b;\theta) + \nabla_\theta c(z)$$

c: Differentiable Neural Networks

RELAX: generalizing control variates w/ neural nets

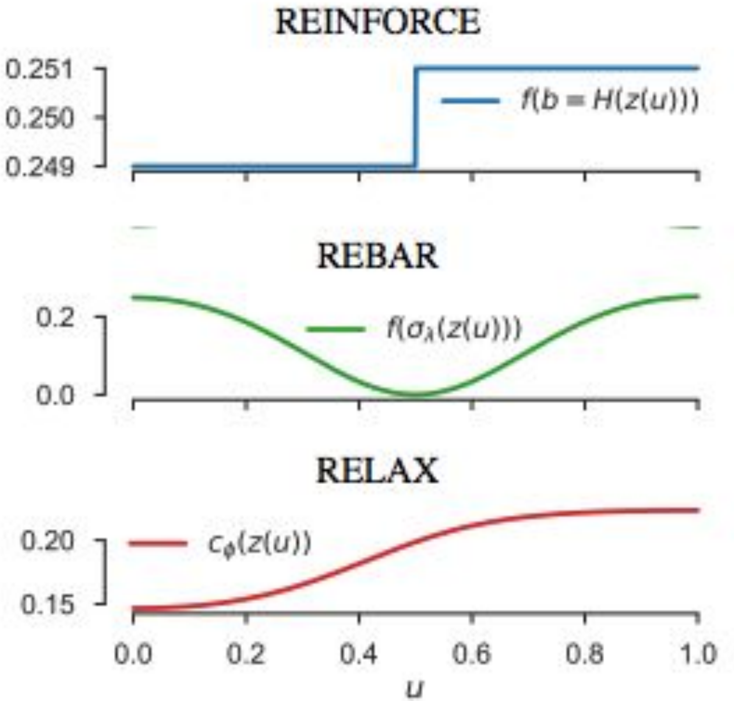

Figure 3: The optimal relaxation for a toy loss function, using different gradient estimators. Because REBAR uses the concrete relaxation of f, which happens to be implemented as a quadratic function, the optimal relaxation is constrained to be a warped quadratic. In contrast, RELAX can choose a free-form relaxation.

Figure from Backpropagation through the Void: Optimizing control variates for black-box gradient estimation, Grathwohl et al., 2017

RELAX: generalizing control variates w/ neural nets

To construct a more powerful gradient estimator, we incorporate a further refinement due to Tucker et al. (2017). Specifically, we evaluate our control variate both at a relaxed input $z \sim p(z|\theta)$, and also at a relaxed input *conditioned on the discrete variable* b, denoted $\tilde{z} \sim p(z|b, \theta)$. Doing so gives us:

$$\hat{g}_{\text{RELAX}} = [f(b) - c_\phi(\tilde{z})] \frac{\partial}{\partial \theta} \log p(b|\theta) + \frac{\partial}{\partial \theta} c_\phi(z) - \frac{\partial}{\partial \theta} c_\phi(\tilde{z}) \tag{8}$$
$$b = H(z), z \sim p(z|\theta), \tilde{z} \sim p(z|b, \theta)$$

This estimator is unbiased for any c_ϕ. A proof and a detailed algorithm can be found in appendix A. We note that the distribution $p(z|b, \theta)$ must also be reparameterizable. We demonstrate how to perform this conditional reparameterization for Bernoulli and categorical random variables in appendix B.

Excerpt from Backpropagation through the Void: Optimizing control variates for black-box gradient estimation, Grathwohl et al., 2017

More Recent Developments

- Credit Assignment in Stochastic Computation Graphs (Weber et al.)
    - generalizing existing techniques under the framework of stochastic computation graph
- Doubly Reparameterized Gradient Estimators (Tucker et al.)
    - fixing high variance gradients in IWAE and more
- Gumbel-sinkhorn (Mena et al.)
    - generalizing Concrete to distribution on permutations
- Implicit reparameterization gradients (Figurnov et al.)
    - broadening the set of distributions we may reparameterize

Summary

- REINFORCE
  - generally applicable (in fact one of the most widely used tricks for RL and NLP!)
  - but is generally of high variance
- Control variates
  - an old idea to reduce variance
  - takes into account a separate *correlated* R.V.
- Reparameterization trick
  - simple reparameterization for multivariate Gaussian
  - generalized reparameterization w/ rejection sampling (**RSVI**)
- Generalizing control variates with
  - concrete variables (**REBAR**)
  - neural networks (**RELAX**)

Other references

- Monte Carlo theory, methods and examples (by Art B. Owen)
  - great book on the theory of Monte Carlo
- Learning Discrete Latent Structure (Duvenaud's course)
  - recently developed techniques for learning discrete structure and their applications
- Differentiable Inference and Generative Models (Duvenaud's course)
  - some *not so recent* stuff on differentiable generative models
- Chris Maddison's Fields institute talk
  - focused on discrete relaxations, video link