Monte Carlo Integrator

Ben Champion
bwc3252@rit.edu

July 15, 2018

1 Basic Algorithm

The integrand $f(x)$ is sampled over a domain V according to a distribution $p(x)$. The Monte Carlo integral over V is

$$I = \frac{V}{N}\sum_{n=0}^{N}\frac{f(x_n)}{p(x_n)}. \tag{1}$$

The variance estimate for this integral is given by

$$Var = \frac{\langle f(x)^2\rangle - \langle f(x)\rangle^2}{N}. \tag{2}$$

In order to reduce this variance without increasing the number of points sampled, the distribution $p(x)$ is chosen to approximate $f(x)$. This is done by iteratively sampling from a Gaussian Mixture Model (GMM), integrating using these samples, and retraining the GMM to better approximate the integrand, stopping either at a predetermined number of iterations or when the variance falls below a certain threshold. When training the GMM each sample is weighted by the ratio of its function value to its responsibility under the current distribution, or

$$w(x_n) = \left|\frac{f(x_n)}{p(x_n)}\right|. \tag{3}$$

Weighting the samples in this way means that the integrator will converge to the correct distribution regardless of how the first iteration is sampled. In practice, the first iteration is sampled uniformly.

2 Initialization

Whether using `monte_carlo_integrator.py` directly or within `mcsampler_new.py`, the integrator is initialized with the following parameters:

1. `d`: Number of dimensions of the integrand's domain

2. `bounds`: `(d × 2)` array where each row is `[left_limit, right_limit]` for its corresponding dimension
3. `gmm_dict`: Python dictionary where each key is a tuple of dimensions and each value is either a sklearn gmm object or `None`
4. `n_comp`: Number of Gaussian components per mixture model

The integrator is called using `monte_carlo_integrator.integrate(func)`, where `func` is a numpy vectorized function. The `err_thresh` and `max_count` parameters can also be specified when calling the integrator.

3 Gaussian Mixture Model

3.1 Basic Gaussian Mixture Model Implementation

The Gaussian mixture model implemented in `gaussian_mixture_model.py` fits a mixture of `k` Gaussians to a set of weighted data using the expectation-maximization (EM) algorithm.

4 Training the GMM

The function `fit_gmm()` fits a Gaussian Mixture Model (GMM) to each group of dimensions specified by `gmm_dict`. Note that in training the GMM, samples drawn from a full, untruncated sampling distribution are used.

The weights used to fit the mixture model are given by

$$weights = \left| \frac{value\_array}{p\_array} \right| \tag{4}$$

where `value_array` is a vertical array of function values for each point, and `p_array` is a vertical array of the responsibility of each point under the current sampling distribution.

For each set of dimensions in `gmm_dict`, the samples for that set of dimensions are placed in a temporary array, and if a corresponding sklearn gmm object does not already exist (i.e. `gmm_dict[dim_group] == None`), one is initialized. The GMM is then trained using `fit(X=samples_to_fit, w=weights)`, and `gmm_dict` is updated with the new GMM.

If sklearn is unable to fit a GMM to the data, `gmm_dict` is updated with `None`.

5 Sampling

5.1 Truncated Multivariate Gaussian Samples

The file `multivariate_truncnorm.py` provides a `sample()` function to draw samples from a multivariate normal distribution inside a set of d-dimensional rectangular bounds. The parameters of `sample()` are the mean and covariance of the distribution, the bounds of the sampling region, and the number of samples to draw.

In general, multivariate Gaussian samples can be generated by sampling from a Gaussian that is centered at the origin with a covariance of 1, applying a linear transformation determined from the covariance matrix, and shifting the resulting samples so that the mean falls in the correct place. For a covariance matrix Σ and mean μ, the final samples are generated as

$$x' = \lambda^{1/2}\phi x + \mu, \tag{5}$$

where x' is the final sample, λ is diagonal matrix of the eigenvalues of Σ, ϕ is a matrix of the eigenvectors of Σ, and x is the original "base sample."

Truncated samples are generated through a similar process. Scipy's `truncnorm` can be used to generate univariate truncated Gaussian samples. Univariate samples with covariance 1 and mean 0 are used to create an array of multivariate samples centered at the origin with covariance 1. The above transformation (equation 5) is used to create truncated samples from the desired distribution.

The problem with this approach, however, is that the bounds are transformed along with the points. This means that the samples fill a d-dimensional parallelogram-shaped region rather than a d-dimensional rectangular one. This problem is remedied by drawing the initial samples from a slightly larger region than is required and rejecting the samples that fall outside the bounds after being transformed.

These new bounds are determined using the inverse of the transformation, or

$$r = [\lambda^{1/2}\phi]^{-1}. \tag{6}$$

Each corner point of the desired sampling region is transformed by r to find the new, transformed corner points. The minimum and maximum coordinates for each dimension describe the smallest rectangular region that, when transformed using equation 5, will contain the desired region.

Unlike traditional rejection sampling, where points are sampled from the entire distribution and then truncated in the end, this method will not suffer drops in efficiency based on where the mean of the distribution is in relation to the bounds. Instead, the limiting factor is the geometry of the transformation.

5.2 Sampling from the GMM

The function `sample_from_gmm()` takes `gmm_dict` and the integrand `func` as parameters. Two empty arrays, `sample_array` (for untruncated samples) and `sample_array_i` (for truncated samples) are initialized. Two arrays of ones, `p_array` and `p_array_i`, are initialized to hold the corresponding sample responsibilities. For each group of dimensions in `gmm_dict`, if `gmm_dict[dim_group]` is a sklearn gmm object, that group of dimensions is sampled as follows:

The lists of means, covariances, and component weights are retrieved from the gmm. For each component, the number of samples to draw is

$$n = N \cdot w, \tag{7}$$

where N is the total number of samples to draw and w is the weight of the component. Using the corresponding mean and covariance matrix, n samples are drawn from an untruncated distribution using numpy's `multivariate_normal` and put in `sample_array`, and n samples are drawn from a truncated distribution using `multivariate_truncnorm` and put in `sample_array_i`. This is done for each component in the current gmm.

The partial responsibilities for both sets of samples are found using sklearn's `score()` function. These responsibilities are multiplied by `p_array` or `p_array_i`, depending on which array of samples they correspond with.

This is repeated for every set of dimensions in `gmm_dict`. The final responsibility for each sample x is

$$p(x) = \prod_{m=0}^{M} p(x_m), \tag{8}$$

where $p(x_m)$ is the responsibility of the group of dimensions m, and M is the total number of groups of dimensions.

If `gmm_dict[dim_group]` has no sklearn gmm object, it is sampled uniformly, and the responsibilities are updated as

$$p(x) = p_{prev.}(x) \prod_{d=0}^{D} \frac{1}{rlim_d - llim_d}, \tag{9}$$

where D is the total number of dimensions in the group of dimensions, and $rlim_d$ and $llim_d$ describe the left and right limits of dimension d, accordingly. This essentially amounds to dividing $p(x)$ by the D-dimensional volume of the region the samples are taken from.

6 Integration

$p(x)$ is the PDF of the current sampling distribution. For an array of samples from $p(x)$ `sample_array`, `k` is the fraction of samples in `sample_array` that fall inside the limits of integration. `sample_array_i` is an array of samples from $p(x)$ that is truncated at the the limits of integration, and `p_array_i` is an array of the responsibility of each point in `sample_array_i` according to $p(x)$. `value_array_i` is the function value at each point in `sample_array_i`. The integral `I` is then

$$I = \frac{k}{N} \sum \frac{value\_array\_i}{p\_array\_i}. \tag{10}$$

The ratio `k` is necessary because $p(x)$ is only normalized to 1 on an infinite domain. The points in `sample_array_i` are sampled from a finite domain, so their responsibilities `p_array_i` must be adjusted by a constant factor `k` to renormalize them.

7 Iterative Algorithm

On the first iteration the integrand is sampled uniformly and `p_array` is an array of ones. This means that the training weights are the function values. The GMM is trained using these samples, and new samples are drawn according to it. The integral and error are calculated (the error is the square root of the variance). The GMM is retrained using the new samples and function values, and this process is iteratively repeated until one or more of the stopping conditions is met.

After calculating the integral and error, if the error is small enough relative to the size of the integral, the integral and variance are added to lists which keep track of each iteration. The running variance is calculated as the weighted average of the variance for each iteration, and the running error is the square root of this variance. The weights are the normalized sums of the responsibilities of the samples for each iteration.

The algorithm stops when either the desired error threshold is met or the maximum number of iterations is reached. To avoid the integrator stopping too early, it will only stop after a minimum number of iterations is reached, regardless of the running error. The final integral is the weighted average of each individual integral, where the weights are once again the sums of the sample responsibilities for each iteration.