Advanced Machine Learning Tutorial Week 1

Benedetta Felici, Jasper Pieterse & Daria Mihaila

November 2023

1 Week 1: Monte Carlo Methods (1)

1.1 Box-Muller method

In order to obtain the joint probability density of $y = (y_1, y_2)$, we can employ the change of variable result, that is:

$$p(y) = p(x = f^{-1}(y)) \left| \det \left(\frac{df}{dx} \right) \right|^{-1} \tag{1}$$

where $p(x) = p(x_1)p(x_2) = 1$ because x_1 and x_2 are independent uniform distributions.

Therefore, we compute the Jacobian:

$$\begin{aligned} \frac{df}{dx} &= \begin{bmatrix} \frac{df_1}{dx_1} & \frac{df_1}{dx_2} \\ \frac{df_1}{dx_2} & \frac{df_2}{dx_2} \end{bmatrix} \\ &= \begin{bmatrix} \frac{cos(2\pi x_2)}{x_1\sqrt{-2log(x_1)}} & -2\pi\sqrt{-2log(x_1)}sin(2\pi x_2) \\ \frac{sin(2\pi x_2)}{x_1\sqrt{-2log(x_1)}} & 2\pi\sqrt{-2log(x_1)}cos(2\pi x_2) \end{bmatrix} \end{aligned} \tag{2}$$

As such, we obtain the following expression for the joint distribution

$$p(y) = \left| det(\frac{df}{dx}) \right|^{-1} = \frac{x_1}{2\pi} \tag{3}$$

From the definition of y_1 and y_2, we derive that

$$y_1^2 + y_2^2 = -2log(x_1)(cos^2(2\pi x_2) + sin^2(2\pi x_2)) = -2log(x_1) \tag{4}$$

.

which gives the following expression for x_1

$$x_1 = \exp - \frac{y_1^2 + y_2^2}{2} \tag{5}$$

Therefore, by using (5) in (3), we obtain:

$$p(y) = \frac{1}{2\pi}\exp-\frac{y_1^2+y_2^2}{2} \quad = \frac{1}{\sqrt{2\pi}}\exp-\frac{y_1^2}{2}\frac{1}{\sqrt{2\pi}}\exp-\frac{y_2^2}{2} = N(y_1|0,1)N(y_2|0,1) \tag{6}$$

Therefore, in order to sample from a normal distribution, with the Box Muller procedure we can sample two values x_1 and x_2 from uniform distributions, and apply f_1 to get y_1, which we have proven to be sampled from a univariate gaussian distribution.

1.2 Diffusion constant scaling

Consider a one-dimensional random walk with steps of size ε to the left or right with equal probability. For T steps, let x_i be the random variable representing the i-th step with $x_i = \varepsilon$ for a step to the right and $x_i = -\varepsilon$ for a step to the left. Thus, $\langle x_i \rangle = 0$ as the step direction is random, with no biases. Therefore, for the mean displacement we expect:

$$\langle X_T \rangle = \langle \sum_{i=1}^{T} x_i \rangle = \sum_{i=1}^{T} \langle x_i \rangle = 0 \tag{7}$$

Where we took the sum outside because the expectation operator is linear. We compute the mean squared displacement as follows:

$$\langle (X_T)^2 \rangle = \langle (\sum_{i=1}^{T} x_i)^2 \rangle = \langle \sum_{i=1}^{T} x_i^2 + \sum_{i\neq j} x_i x_j \rangle = \langle \sum_{i=1}^{T} x_i^2 \rangle = T\epsilon^2 \tag{8}$$

Where we used that $x_i^2 = \epsilon^2$ regardless what step is taken and $\langle x_i x_j \rangle = 0$ as there is no correlation between the steps. So, for the root mean squared displacement, representing the distance travelled after T steps is:

$$X_{rms} = \sqrt{\langle X_T^2 \rangle} = \sqrt{T}\epsilon \tag{9}$$

Therefore, after T steps, the state is likely to have moved a distance about $\sqrt{T}\varepsilon$ and the diffusion (rate at which particles spread out in a random walk) scales as $\sqrt{T}$. For this reason, we usually want to avoid random walk behaviour in our MCMC simulations, they take very long to converge.

1.3 A study of importance sampling

Analytical Derivation

We want to know the variance of the importance weights when estimating a statistic Φ of a distribution $P(x) \sim \mathcal{N}(0, \sigma_p)$ using importance sampling under distribution $Q(x) \sim \mathcal{N}(0, \sigma_q)$ with

$$\Phi = \int dx\, P(x)\phi(x), \tag{10}$$

Where we assume that $P^* = P$, so P is actually normalized, and $Z = 1$.

The weight $w(x)$ for a point x drawn from Q is given by:

$$w(x) = \frac{P(x)}{Q(x)} = \frac{\frac{1}{\sqrt{2\pi}\sigma_p}\exp\left(-\frac{x^2}{2\sigma_p^2}\right)}{\frac{1}{\sqrt{2\pi}\sigma_q}\exp\left(-\frac{x^2}{2\sigma_q^2}\right)} = \frac{\sigma_q}{\sigma_p}\exp\left(-\frac{x^2}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma_q^2}\right)\right) \tag{11}$$

The variance of the weights (drawn from $Q(x)$) is:

$$\mathrm{Var}(w) = \langle w^2 \rangle - \langle w \rangle^2$$

With

$$\langle w^2 \rangle = \int dx\, Q(x) w(x)^2$$

Substituting the expression for $w(x)$ and $Q(x)$, we get

$$\langle w^2 \rangle = \int dx\, \frac{1}{\sqrt{2\pi}\sigma_q}\exp\left(-\frac{x^2}{2\sigma_q^2}\right)\frac{\sigma_q^2}{\sigma_p^2}\left(\exp\left(-\frac{x^2}{2}\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma_q^2}\right)\right)\right)^2$$

This simplifies to

$$\begin{aligned}\langle w^2 \rangle &= \frac{1}{\sqrt{2\pi}\sigma_q}\frac{\sigma_q^2}{\sigma_p^2}\int dx\, \exp\left(-\frac{x^2}{2\sigma_q^2}\right)\exp\left(-x^2\left(\frac{1}{\sigma_p^2} - \frac{1}{\sigma_q^2}\right)\right) \\ &= \frac{\sigma_q}{\sqrt{2\pi}\sigma_p^2}\int dx\, \exp\left(-x^2\left(\frac{1}{\sigma_p^2} - \frac{1}{2\sigma_q^2}\right)\right)\end{aligned}$$

We also have:

$$\langle w \rangle = \int dx\, Q(x) w(x) = \int dx\, Q(x)\frac{P(x)}{Q(x)} = 1$$

Since P is normalized and $Z = 1$

Then the variance of the weights is given by

$$\mathrm{Var}(w) = \langle w^2 \rangle - \langle w \rangle^2 = \frac{\sigma_q}{\sqrt{2\pi}\sigma_p^2}\int dx\, \exp\left(-x^2\left(\frac{1}{\sigma_p^2} - \frac{1}{2\sigma_q^2}\right)\right) - 1$$

Using the Gaussian integral, this further simplifies to

$$\mathrm{Var}(w) = \frac{\sigma_q}{\sigma_p^2\sqrt{\frac{2}{\sigma_p^2} - \frac{1}{\sigma_q^2}}} - 1$$

From this expression, we see that as

$$\lim_{\sigma_q^2 \to \frac{\sigma_p^2}{2}} \mathrm{Var}(w) = \infty$$

. Furthermore, once

$$\sigma_q^2 < \frac{\sigma_p^2}{2}$$

the expression becomes invalid.

Computational Simulation

We set $\sigma_p = 1$, such that $\sigma_q > \sqrt{\frac{1}{2}}$, varying it from 0.1 up to 1.6 in our calculations. We also set the numbers of samples as in figure 29.20 so $N_{values} = [1000, 10000, 100000]$ and compute each part using the same seed, for generating smooth curves.
We replicated the figure 29.20 as evidenced below:

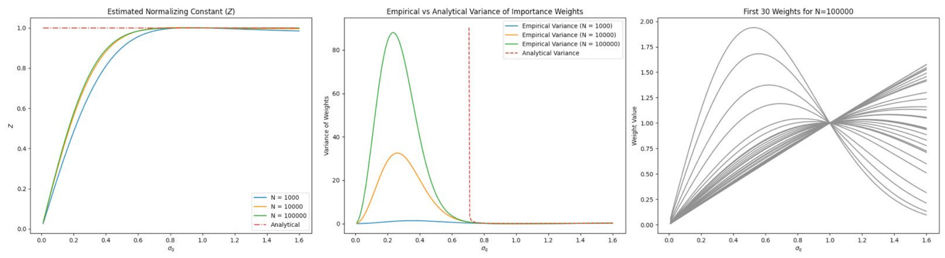

Figure 1: The three plots show (a) the estimated normalizing constant; (b) the empirical standard deviation of the R weights; (c) 30 of the weights.

For a), for each σ_q and N_{value}, we compute the weights and then take their average, giving us the estimated / empirical normalizing constant. In b) we take samples from σ_q, calculate the weights for each and take their variances. Lastly, for c), we will plot the first 30 weights for N=100000.

This is our code for generating the figures:

```
import numpy as np
import matplotlib.pyplot as plt

# set parameters
sigma_p = 1
sigma_q_values_anal = np.linspace(np.sqrt(.500015), 1.6, 200) # ;)
sigma_q_values_emp = np.linspace(.01, 1.6, 200)
N_values = [1000, 10000, 100000] # as in figure 29.20

# calculate Z_values and store weights
Z_values = np.zeros((len(sigma_q_values_emp), len(N_values)))
weights_100k = np.zeros((len(sigma_q_values_emp), 30)) # storing first 30 weights
for each sigma_q

for i, sigma_q in enumerate(sigma_q_values_emp):
    for j, N in enumerate(N_values):
        np.random.seed(20000)
        sampled_points = np.random.normal(0, sigma_q, N)
        p = (1 / np.sqrt(2 * np.pi)) * np.exp(-sampled_points**2 / 2)
        q = (1 / np.sqrt(2 * np.pi * sigma_q**2)) * np.exp(-sampled_points**2 /
        (2 * sigma_q**2))
        w = p / q
        Z_values[i, j] = np.mean(w)
        if N == 100000:
            weights_100k[i] = w[:30]

# function to calculate empirical weights
def weight_vectorized(samples, sigma_p, sigma_q):
    return (sigma_q / sigma_p) * np.exp(-samples**2 / 2 * (1/sigma_p**2 -
    1/sigma_q**2))

# function to calculate analytical variance
def analytical_variance(sigma_p, sigma_q):
    if 2/sigma_p**2 <= 1/sigma_q**2:
        raise ValueError("Invalid standard deviations: ensure that sigma_q^2 >
        sigma_p^2 / 2")
    variance = (sigma_q / (sigma_p**2 * np.sqrt(2/sigma_p**2 - 1/sigma_q**2))) - 1
    return variance

# reate subplots
fig, axs = plt.subplots(1, 3, figsize=(22, 6))
```

```
# Z-values plot
for i, N in enumerate(N_values):
    axs[0].plot(sigma_q_values_emp, Z_values[:, i], label=f'N = {N}')
ones = np.ones(sigma_q_values_emp.shape[0])
axs[0].plot(sigma_q_values_emp, ones, label='Analytical', linestyle='-.')
axs[0].set_xlabel(r'$\sigma_q$')
axs[0].set_ylabel('$Z$')
axs[0].set_title('Estimated Normalizing Constant ($Z$)')
axs[0].legend()

# empirical and analytical variance plot
for N in N_values:
    empirical_variances = []
    for sigma_q in sigma_q_values_emp:
        np.random.seed(66)
        samples_from_q = np.random.normal(0, sigma_q, N)
        weights = weight_vectorized(samples_from_q, sigma_p, sigma_q)
        empirical_variances.append(np.var(weights))
    axs[1].plot(sigma_q_values_emp, empirical_variances, label=f'Empirical
    Variance (N = {N})')
analytical_variances = [analytical_variance(sigma_p, sigma_q) for sigma_q in
sigma_q_values_anal]
axs[1].plot(sigma_q_values_anal, analytical_variances, label='Analytical
Variance', linestyle='--')
axs[1].set_xlabel(r'$\sigma_q$')
axs[1].set_ylabel('Variance of Weights')
axs[1].set_title('Empirical vs Analytical Variance of Importance Weights')
axs[1].legend()

# plot of the first 30 weights for N=100000
for i in range(30):
    axs[2].plot(sigma_q_values_emp, weights_100k[:, i], color = 'grey')
axs[2].set_xlabel(r'$\sigma_q$')
axs[2].set_ylabel('Weight Value')
axs[2].set_title('First 30 Weights for N=100000')

# show plots
plt.tight_layout()
plt.show()
```