Mathematical Foundations of Deep Neural Networks, M1407.001200
E. Ryu
Spring 2024

Homework 10
Due 5pm, Monday, May 27, 2024

Problem 1: *Log-derivative trick for VAE.* Let $Z \in \mathbb{R}^k$ be a random variable. Let $q_\phi(\cdot)$ be a probability density function for all $\phi \in \mathbb{R}^p$. Assume $q_\phi(z)$ is differentiable in ϕ for all fixed $z \in \mathbb{R}^k$. Let $h\colon \mathbb{R}^k \to \mathbb{R}$ satisfy $h(z) > 0$ for all $z \in \mathbb{R}^k$. Assume that the order of integration and differentiation can be swapped. Show

$$\nabla_\phi \mathbb{E}_{Z\sim q_\phi}\left[\log\left(\frac{h(Z)}{q_\phi(Z)}\right)\right] = \mathbb{E}_{Z\sim q_\phi}\left[(\nabla_\phi \log q_\phi(Z))\log\left(\frac{h(Z)}{q_\phi(Z)}\right)\right].$$

Hint. Since $q_\phi(z)$ is a probability density function,

$$\int \nabla_\phi q_\phi(z)\,dz = \nabla_\phi \int q_\phi(z)\,dz = \nabla_\phi 1 = 0.$$

Solution. Observe

$$\nabla_\phi \mathbb{E}_{Z\sim q_\phi}\left[\log\left(q_\phi(Z)\right)\right] = \nabla_\phi \int \log\left(q_\phi(z)\right) q_\phi(z)\,dz = \int \left(1 + \log\left(q_\phi(z)\right)\right)\nabla_\phi q_\phi(z)\,dz.$$

In the last equation, we've used the product rule and the fact $\nabla_\phi \log\left(q_\phi(z)\right) = \frac{\nabla_\phi q_\phi(z)}{q_\phi(z)}$. Applying above equation we have

$$\begin{aligned}\nabla_\phi \mathbb{E}_{Z\sim q_\phi}\left[\log\left(\frac{h(Z)}{q_\phi(Z)}\right)\right] &= \nabla_\phi\left(\mathbb{E}_{Z\sim q_\phi}\left[\log\left(h(Z)\right)\right] - \mathbb{E}_{Z\sim q_\phi}\left[\log\left(q_\phi(Z)\right)\right]\right)\\ &= \nabla_\phi \mathbb{E}_{Z\sim q_\phi}\left[\log\left(h(Z)\right)\right] - \int \left(1 + \log\left(q_\phi(z)\right)\right)\nabla_\phi q_\phi(z)\,dz.\end{aligned}$$

Using the log-derivative trick, we have

$$\nabla_\phi \mathbb{E}_{Z\sim q_\phi}\left[\log\left(h(Z)\right)\right] = \mathbb{E}_{Z\sim q_\phi}\left[\log\left(h(Z)\right)\nabla_\phi \log q_\phi(Z)\right].$$

Next, as in the hint, $\int \nabla_\phi q_\phi(z)\,dz = 0$. Lastly,

$$\begin{aligned}\int \log\left(q_\phi(z)\right)\nabla_\phi q_\phi(z)\,dz &= \int \left[\log\left(q_\phi(z)\right)\nabla_\phi \log(q_\phi(z))\right] q_\phi(z)\,dz\\ &= \mathbb{E}_{Z\sim q_\phi}\left[\log\left(q_\phi(Z)\right)\nabla_\phi \log(q_\phi(Z))\right].\end{aligned}$$

Thus,

$$\begin{aligned}&\nabla_\phi \mathbb{E}_{Z\sim q_\phi}\left[\log\left(\frac{h(Z)}{q_\phi(Z)}\right)\right]\\ =&\mathbb{E}_{Z\sim q_\phi}\left[\log\left(h(Z)\right)\nabla_\phi \log q_\phi(Z)\right] - \mathbb{E}_{Z\sim q_\phi}\left[\log\left(q_\phi(Z)\right)\nabla_\phi \log(q_\phi(Z))\right]\\ =&\mathbb{E}_{Z\sim q_\phi}\left[\log\left(\frac{h(Z)}{q_\phi(Z)}\right)\nabla_\phi \log(q_\phi(Z))\right].\end{aligned}$$

■

Problem 2: *Projected gradient method.* Consider the optimization problem

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & f(x) \\ \text{subject to} & x \in C, \end{array}$$

where $C \subset \mathbb{R}^n$. Constrained optimization problems of this type can be solved with the *projected gradient method*

$$x^{k+1} = \Pi_C(x^k - \alpha \nabla f(x^k)),$$

where Π_C is the projection onto C. The projection of $y \in \mathbb{R}^n$ onto $C \subseteq \mathbb{R}^n$ is defined as the point in C that is closest to y:

$$\Pi_C(y) = \underset{x\in C}{\operatorname{argmin}} \|x - y\|^2.$$

For the particular set

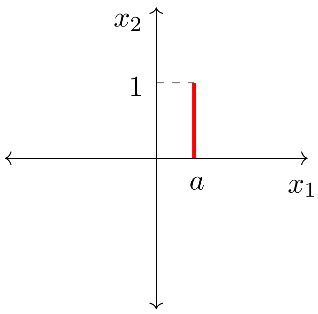
$$C = \{x \in \mathbb{R}^2 \,|\, x_1 = a,\, 0 \le x_2 \le 1\},$$

where $a \in \mathbb{R}$, show that

$$\Pi_C(y) = \begin{bmatrix} a \\ \min\{\max\{y_2, 0\}, 1\} \end{bmatrix},$$

where $y = (y_1, y_2)$.

Solution.



First, considering each case, we can check

$$\min\{\max\{y_2, 0\}, 1\} = \begin{cases} 1 & \text{if } y_2 > 1 \\ y_2 & \text{if } 0 \le y_2 \le 1 \\ 0 & \text{if } y_2 < 0. \end{cases}$$

Considering the same case division, we can check

$$\underset{0\le x_2\le 1}{\operatorname{argmin}} (x_2 - y_2)^2 = \min\{\max\{y_2, 0\}, 1\}.$$

Therefore, for $y = (y_1, y_2)$ we conclude

$$\begin{aligned} \Pi_C(y) &= \underset{x\in C}{\operatorname{argmin}} \|x - y\|^2 = \underset{(a,x_2)\in C}{\operatorname{argmin}} \left[(a - y_1)^2 + (x_2 - y_2)^2\right] \\ &= \begin{bmatrix} a \\ \operatorname{argmin}_{0\le x_2\le 1}(x_2 - y_2)^2 \end{bmatrix} = \begin{bmatrix} a \\ \min\{\max\{y_2, 0\}, 1\} \end{bmatrix}. \end{aligned}$$

■

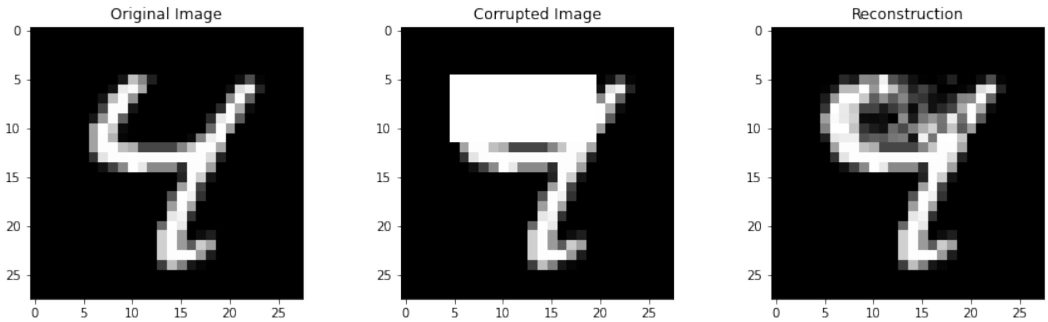


Figure 1: The original, corrupted, and inpainted MNIST image.

Problem 3: *Image inpainting with flow models.* Assume we have a trained flow model that we use to evaluate the likelihood function p. (Since we will not further train or update the flow model, we supress the network parameter θ and write p rather than p_θ.) The starter code `flow_inpainting.py` loads a NICE flow model pre-trained on the MNIST dataset saved in `nice.pt`. Let $X_{\text{true}} \in \mathbb{R}^{28\times 28}$ be an MNIST image with pixel intensities normalized to be in $[0,1]$. Let $M = \{0,1\}^{28\times 28}$ be a binary mask. We measure $M \odot X_{\text{true}}$, where $\odot$ denotes elementwise multiplication, and the goal is to inpaint the missing information $(1-M)\odot X_{\text{true}}$, where $1-M \in \{0,1\}^{28\times 28}$ is the inverted mask. (See Figure 1.) Perform inpainting by solving the following constrained maximum likelihood estimation problem

$$\begin{array}{ll} \underset{X\in\mathbb{R}^{28\times 28}}{\text{minimize}} & -\log p(X) \\ \text{subject to} & M\odot X = M\odot X_{\text{true}} \\ & 0 \le X \le 1, \end{array}$$

where $0 \le X \le 1$ is enforced elementwise. Use the projected gradient method with learning rate 10^{-3} and 300 iterations.

Hint. Represent the optimization variable with

```
X = image.clone().requires_grad_(True)
```

while preserving `image`, the tensor containing the corrupted image. When manipulating `X` in the projection step, manipulate `X.data` rather than `X` itself so that the computation graph is not altered by the projection step. Use `clamp(...)` to enforce the $0 \le X \le 1$ constraint.

Remark. The optimization problem can be interpreted as finding the most likely reconstruction consistent with the measurements.

Remark. The NICE paper [2] obtains better inpainting results by using a learning rate scheduler (iteration-dependent stepsize) and adding noise to escape from local minima.

Solution. See `flow_inpainting_sol.py`.

Problem 4: *Ingredients of Glow [1].* Let

$$A = PL(U + \operatorname{diag}(s)) \in \mathbb{R}^{C\times C},$$

where $P \in \mathbb{R}^{C\times C}$ is a permutation matrix, $L \in \mathbb{R}^{C\times C}$ is a lower triangular matrix with unit diagonals, $U \in \mathbb{R}^{C\times C}$ is upper triangular with zero diagonals, and $s \in \mathbb{R}^C$. To clarify, $L_{ii} = 1$ for $i = 1, \ldots, C$, $L_{ij} = 0$ for $1 \le i < j \le C$, and $U_{ij} = 0$ for $1 \le j \le i \le C$.

(a) Let $f_1(x) = Ax$. Show

$$\log\left|\frac{\partial f_1}{\partial x}\right| = \sum_{i=1}^{C} \log|s_i|.$$

(b) Given $h\colon \mathbb{R}^{a\times b\times c} \to \mathbb{R}^{a\times b\times c}$, define

$$\left|\frac{\partial h(X)}{\partial X}\right| = \left|\frac{\partial(h(X).\text{reshape}(abc))}{\partial(X.\text{reshape}(abc))}\right|,$$

i.e., we define the absolute value of the Jacobian determinant with the input and output tensors vectorized. Note that the reshape operation, which maps elements from the tensor in $\mathbb{R}^{a\times b\times c}$ to the elements of the vector in $\mathbb{R}^{abc}$, is not unique. Show that the definition of $\left|\frac{\partial h(X)}{\partial X}\right|$ does not depend on the specific choice of reshape.

(c) Let $f_2(X \mid P, L, U, s)$ be the 1×1 convolution from $\mathbb{R}^{C\times m\times n}$ to $\mathbb{R}^{C\times m\times n}$ with filter $w \in \mathbb{R}^{C\times C\times 1\times 1}$ defined as

$$w_{i,j,1,1} = A_{i,j}, \qquad \text{for } i = 1, \ldots, C,\ j = 1, \ldots, C.$$

So $X \in \mathbb{R}^{C\times m\times n}$ and $f_2(X \mid P, L, U, s) \in \mathbb{R}^{C\times m\times n}$. (Assume the batch size is 1.) Show

$$\log\left|\frac{\partial f_2(X \mid P, L, U, s)}{\partial X}\right| = mn\sum_{i=1}^{C} \log|s_i|.$$

(d) Consider the following coupling layer from $X \in \mathbb{R}^{2C\times m\times n}$ to $Z \in \mathbb{R}^{2C\times m\times n}$:

$$\begin{aligned} Z_{1:C,:,:} &= X_{1:C,:,:} \\ Z_{C+1:2C,:,:} &= f_2(X_{C+1:2C,:,:}|P, L(X_{1:C,:,:}), U(X_{1:C,:,:}), s(X_{1:C,:,:})), \end{aligned}$$

where P is a fixed permutation matrix, $L(\cdot)$ outputs lower triangular matrices with unit diagonals in $\mathbb{R}^{C\times C}$, $U(\cdot)$ outputs upper triangular matrices with zero diagonals in $\mathbb{R}^{C\times C}$, and $s(\cdot) \in \mathbb{R}^C$. Show

$$\log\left|\frac{\partial Z}{\partial X}\right| = mn\sum_{i=1}^{C} \log|s_i|.$$

Remark. Given any $A \in \mathbb{R}^{n\times n}$, a decomposition $A = PL(U + \operatorname{diag}(s))$ can be computed via the so-called PLU factorization, which performs steps analogous to Gaussian elimination.

Solution.

(a) First, derivative of linear transformation is the matrix multiplied with the input. Thus

$$\left|\frac{\partial f_1}{\partial x}\right| = |\det(A)|\,.$$

Next, determinant can be split into its multiplicative factors

$$|\det(A)| = |\det(P)\det(L)\det(U + \operatorname{diag}(s))|\,.$$

Lastly, as the determinant of a triangular matrix is the product of its diagonal entries and the determinant of a permutation is ± 1, we see

$$|\det(P)| = 1, \quad |\det(L)| = 1, \quad |\det(U + \operatorname{diag}(s))| = \prod_{i=1}^{C} |s_i|.$$

Therefore

$$|\det(A)| = \prod_{i=1}^{C} |s_i|,$$

taking the logarithm of both sides, we get the desired conclusion.

(b) For different choices of reshape, we may consider them as functions that map elements from the tensor in $\mathbb{R}^{a\times b\times c}$ to the elements of the vector in $\mathbb{R}^{abc}$. Let r_1 and r_2 be the functions correspond to two different choices of reshape. We want to show

$$\left|\frac{\partial r_1(h(X))}{\partial r_1(X)}\right| = \left|\frac{\partial r_2(h(X))}{\partial r_2(X)}\right|.$$

As the outputs of r_1 and r_2 are consisted with the same entries with different order, there is a permutation matrix $P\colon \mathbb{R}^{abc} \to \mathbb{R}^{abc}$ that satisfies $r_1(X) = Pr_2(X)$ for all tensors X in $\mathbb{R}^{a\times b\times c}$. Applying the chain rule, we have

$$\left|\frac{\partial r_1(h(X))}{\partial r_1(X)}\right| = \left|\frac{\partial Pr_2(h(X))}{\partial Pr_2(X)}\right| = \left|\frac{\partial Pr_2(h(X))}{\partial r_2(h(X))}\right|\left|\frac{\partial r_2(h(X))}{\partial r_2(X)}\right|\left|\frac{\partial r_2(X))}{\partial Pr_2(X)}\right|.$$

However, as P is a permutation matrix, we know

$$\left|\frac{\partial Pr_2(h(X))}{\partial r_2(h(X))}\right| = |\det(P)| = 1, \qquad \left|\frac{\partial r_2(X)}{\partial Pr_2(X)}\right| = |\det(P^{-1})| = 1.$$

Plugging to the previous equation, we conclude the desired result.

(c) We consider the function r that corresponds to a reshape operation that satisfies

$$X_{\gamma,a,b} = r(X)_{(a-1)nC+(b-1)C+\gamma}.$$

From the definition of f_2, for $1 \le a \le m$, $1 \le b \le n$ and $1 \le \ell, \gamma \le C$ we have

$$f_2(X \mid P, L, U, s)_{\ell,a,b} = \sum_{\gamma=1}^{C} w_{\ell,\gamma,1,1} X_{\gamma,a,b} = \sum_{\gamma=1}^{C} A_{\ell,\gamma} X_{\gamma,a,b}.$$

As the bias does not change the derivative, we assumed the bias to be zero for the sake of simplicity. From above equation, (under the identification by r) we have

$$\frac{\partial f_2(X \mid P, L, U, s)_{\ell,a',b'}}{\partial X_{\gamma,a,b}} = \begin{cases} A_{\ell,\gamma} & \text{if } a = a', b = b' \\ 0 & \text{otherwise.} \end{cases}$$

From (b) we know the absolute value of the Jacobian determinant does not depend on the specific choice of reshape, therefore

$$\left| \frac{\partial f_2(X \mid P, L, U, s)}{\partial X} \right| = \left| \frac{\partial r(f_2(X \mid P, L, U, s))}{\partial r(X)} \right| = \left| \det(\operatorname{diag}(A, A \dots, A)) \right|,$$

where $\operatorname{diag}(A, A, \dots, A)$ is an $mn \times mn$ block diagonal matrix with diagonal entries A. On the other hand, from (a) we know $|\det(A)| = \prod_{i=1}^{C} |s_i|$, therefore

$$\left| \det(\operatorname{diag}(A, A \dots, A)) \right| = \prod_{a=1}^{m} \prod_{b=1}^{n} |\det(A)| = |\det(A)|^{mn} = \left(\prod_{i=1}^{C} |s_i| \right)^{mn}.$$

Finally, taking the logarithm of both sides, we get the desired conclusion.

(d) Observe, for $1 \leq a \leq m$ and $1 \leq b \leq n$ we have

$$\begin{aligned} \left| \frac{\partial Z_{1:C,a,b}}{\partial X_{1:C,a,b}} \right| &= \left| \det(I) \right|, \\ \left| \frac{\partial Z_{C+1:2C,a,b}}{\partial X_{C+1:2C,a,b}} \right| &= \left| \det(A) \right|, \\ \left| \frac{\partial Z_{1:C,a,b}}{\partial X_{C+1:2C,a,b}} \right| &= 0. \end{aligned}$$

In this context, we have

$$\left| \frac{\partial Z_{:,a,b}}{\partial X_{:,a,b}} \right| = \left| \det \begin{pmatrix} I & 0 \\ D & A \end{pmatrix} \right| = \left| \det(I) \det(A) \right| = \left| \det(A) \right|,$$

here D is a $C \times C$ matrix that satisfies $|\det(D)| = \left| \frac{\partial Z_{C+1:2C,a,b}}{\partial X_{1:C,a,b}} \right|$.

Now, with the similar argument of (c) we have

$$\left| \frac{\partial Z}{\partial X} \right| = \prod_{a=1}^{m} \prod_{b=1}^{n} |\det(A)| = \left(\prod_{i=1}^{C} |s_i| \right)^{mn},$$

and taking the logarithm of both sides, we get the desired conclusion.

■

Problem 5: *Gambler's ruin.* You are a gambler at a casino with a starting balance of 100\$. You will play a game in which you bet 1\$ every game. With probability 18/37, you win and collect 2\$ (so you make a 1\$ profit). With probability 19/37, you lose and collect no money. You play until you reach a balance of 0\$ or 200\$ or until you play 600 games. Write a Monte Carlo simulation with importance sampling to estimate the probability that you leave the casino with 200\$. Specifically, simulate playing up to 600 games until you reach the balance of 0\$ or 200\$ and repeat this $N = 3000$ times.

Hint. Regardless of the outcome, simulate $K = 600$ games. The outcomes of the games form a sequence of Bernoulli random variables with probability mass function

$$f(X_1, \dots, X_K) = \prod_{i=1}^{K} p^{X_i}(1-p)^{(1-X_i)}$$

and $p = 18/37$. For the sampling distribution, also use a sequence of Bernoulli random variables with probability mass function

$$g(Y_1, \dots, Y_K) = \prod_{i=1}^{K} q^{Y_i}(1-q)^{(1-Y_i)}$$

but with $q > p$. Try using $q = 0.55$.

Hint. The answer is approximately 2×10^{-6}. Submit Python code that produces this answer.

Solution. See `prob5.py`. ■

Problem 6: Solve

$$\begin{array}{ll} \underset{\mu,\sigma\in\mathbb{R}}{\text{minimize}} & \mathbb{E}_{X\sim\mathcal{N}(\mu,\sigma^2)}[X\sin(X)] + \frac{1}{2}(\mu-1)^2 + \sigma - \log\sigma \\ \text{subject to} & \sigma > 0 \end{array}$$

using SGD combined with

(a) the log-derivative trick and

(b) the reparameterization trick.

Hint. Use the change of variables $\sigma = e^\tau$ to remove the constraint $\sigma > 0$.

Clarification. Implement SGD in Python and submit the code.

Solution. To remove the constraint, apply change of variables $\sigma = e^\tau$, then the equivalent problem is

$$\underset{\mu,\tau\in\mathbb{R}}{\text{minimize}} \quad \mathbb{E}_{X\sim\mathcal{N}(\mu,e^{2\tau})}[X\sin(X)] + \frac{1}{2}(\mu-1)^2 + e^\tau - \tau.$$

(a) First, use the log-derivative. The log-pdf of $\mathcal{N}(\mu, e^{2\tau})$ is

$$\log f(x, \mu, e^{2\tau}) = -\frac{1}{2}\log(2\pi) - \tau - \frac{1}{2}\frac{(x-\mu)^2}{e^{2\tau}}.$$

The derivative of log-pdf of $\mathcal{N}(\mu, e^{2\tau})$ is

$$\begin{aligned} \nabla_\mu \log f(x, \mu, e^{2\tau}) &= \frac{x-\mu}{e^{2\tau}} \\ \nabla_\tau \log f(x, \mu, e^{2\tau}) &= -1 + \frac{(x-\mu)^2}{e^{2\tau}}. \end{aligned}$$

Thus, the gradient of the given problem is

$$\begin{aligned}
&\nabla_\mu \left(\mathbb{E}_{X\sim\mathcal{N}(\mu,e^{2\tau})}[X\sin(X)] + \frac{1}{2}(\mu-1)^2 + e^\tau - \tau\right) \\
&\quad = \mathbb{E}_{X\sim\mathcal{N}(\mu,e^{2\tau})}\left[X\sin(X)\frac{X-\mu}{e^{2\tau}}\right] + \mu - 1 \\
&\nabla_\tau \left(\mathbb{E}_{X\sim\mathcal{N}(\mu,e^{2\tau})}[X\sin(X)] + \frac{1}{2}(\mu-1)^2 + e^\tau - \tau\right) \\
&\quad = \mathbb{E}_{X\sim\mathcal{N}(\mu,e^{2\tau})}\left[X\sin(X)\left(-1 + \frac{(X-\mu)^2}{e^{2\tau}}\right)\right] + e^\tau - 1.
\end{aligned}$$

For the SGD, perform

$$\begin{aligned}
&X_1, X_2, ..., X_B \sim \mathcal{N}(\mu^k, e^{2\tau^k}) \\
&\mu^{k+1} = \mu^k - \alpha\left(\frac{1}{B}\sum_{i=1}^{B} X_i\sin(X_i)\frac{X_i-\mu^k}{e^{2\tau^k}} + \mu^k - 1\right) \\
&\tau^{k+1} = \tau^k - \alpha\left(\frac{1}{B}\sum_{i=1}^{B} X_i\sin(X_i)\left(-1 + \frac{(X_i-\mu^k)^2}{e^{2\tau^k}}\right) + e^{\tau^k} - 1\right).
\end{aligned}$$

(b) Now, for the reparameterization trick, consider

$$\mathbb{E}_{X\sim\mathcal{N}(\mu,e^{2\tau})}[X\sin(X)] = \mathbb{E}_{Z\sim\mathcal{N}(0,1)}[(e^\tau Z + \mu)\sin(e^\tau Z + \mu)].$$

Thus, the gradient of the given problem is

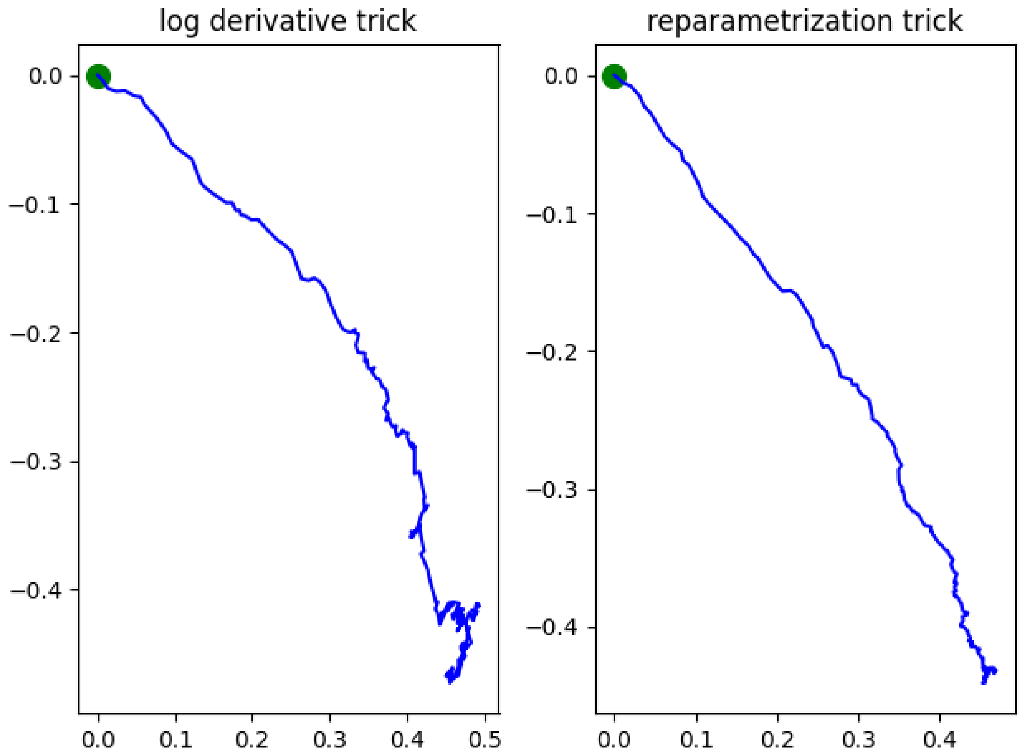
$$\begin{aligned}
&\nabla_\mu \left(\mathbb{E}_{X\sim\mathcal{N}(\mu,e^{2\tau})}[X\sin(X)] + \frac{1}{2}(\mu-1)^2 + e^\tau - \tau\right) \\
&\quad = \mathbb{E}_{Z\sim\mathcal{N}(0,1)}[(e^\tau Z + \mu)\cos(e^\tau Z + \mu) + \sin(e^\tau Z + \mu)] + \mu - 1 \\
&\nabla_\tau \left(\mathbb{E}_{X\sim\mathcal{N}(\mu,e^{2\tau})}[X\sin(X)] + \frac{1}{2}(\mu-1)^2 + e^\tau - \tau\right) \\
&\quad = \mathbb{E}_{Z\sim\mathcal{N}(0,1)}\left[e^\tau Z\left(\sin(e^\tau Z + \mu) + (e^\tau Z + \mu)\cos(e^\tau Z + \mu)\right)\right] + e^\tau - 1.
\end{aligned}$$

For the SGD, perform

$$\begin{aligned}
&Z_1, Z_2, ..., Z_B \sim \mathcal{N}(0,1) \\
&\mu^{k+1} = \mu^k - \alpha\left(\frac{1}{B}\sum_{i=1}^{B}\left((e^{\tau^k}Z_i + \mu^k)\cos(e^{\tau^k}Z_i + \mu^k) + \sin(e^{\tau^k}Z_i + \mu^k)\right) + \mu^k - 1\right) \\
&\tau^{k+1} = \tau^k - \alpha\left(\frac{1}{B}\sum_{i=1}^{B} e^{\tau^k}Z_i\left(\sin(e^{\tau^k}Z_i + \mu^k) + (e^{\tau^k}Z_i + \mu^k)\cos(e^{\tau^k}Z_i + \mu^k)\right) + e^{\tau^k} - 1\right).
\end{aligned}$$

The implementation of SGD by python is done in the `prob6.py` file. Following figure is the path by (μ, τ) using log-derivative trick and reparameterization trick.

■



References

[1] D. P. Kingma and P. Dhariwal, Glow: Generative flow with invertible 1x1 convolutions, *NeurIPS*, 2018.

[2] L. Dinh, D. Krueger, and Y. Bengio, NICE: Non-linear independent components estimation, *ICLR Workshop*, 2015.