Sample-Efficient Optimization in the Latent Space of Deep Generative Models via Weighted Retraining

- 作者：Austin Tripp，Erik Daxberger，José Miguel Hernández-Lobato
- 机构：University of Cambridge，Microsoft Research

- 会议：NIPS2020
- 地址：https://arxiv.org/abs/2006.09191
- 代码：https://github.com/cambridge-mlg/weighted-retraining

论文主要内容

摘要

许多实际问题涉及在复杂、高维、结构化(graph、sequence、sets)空间内优化一个昂贵的黑盒函数。机器学习方法对这样的问题很有效，但是已有的方法非常缺少sample efficgiency。本文提出使用一个deep generative model来将优化的空间变成低维、连续的latent manifold。通过在优化迭代中周期***retaining*** generative model、***weighting*** data points实现。

研究内容

Motivation

- *Latent space optimization*（LSO）方法
    - 1: 构建一个(deep)生成模型，用来将低维连续隐空间的tensors映射到输入空间的数据manifold
    - 2: 在latent space上优化目标函数（使用surrogate model）
    - 问题：
        - 一般生成模型使用pretrained，没有针对下游任务专门精心定制（从优化任务中decouple)。
        - 给优化造成不必要的困难
- 基于此提出：对数据分布weighting、周期retaining生成模型来消除decouple

方法

介绍

背景

Sample-Efficient Black Box Optimization

- $\mathcal{X}$ 是input space（这篇文章针对于高维的、结构化的）
- $f : \mathcal{X} \mapsto \mathbb{R}$ 是优化的目标函数。 f 是黑盒的，即不知道解析式、无法获得梯度信息。evaluate的代价很高(昂贵)
- problem：通过evaluate f 次数尽量少优化。 $\mathcal{D}_M \equiv \{\mathbf{x}_i, f(\mathbf{x}_i)\}_{i=1}^M$ 为evalute M次的一个序列

Model-based Optimization

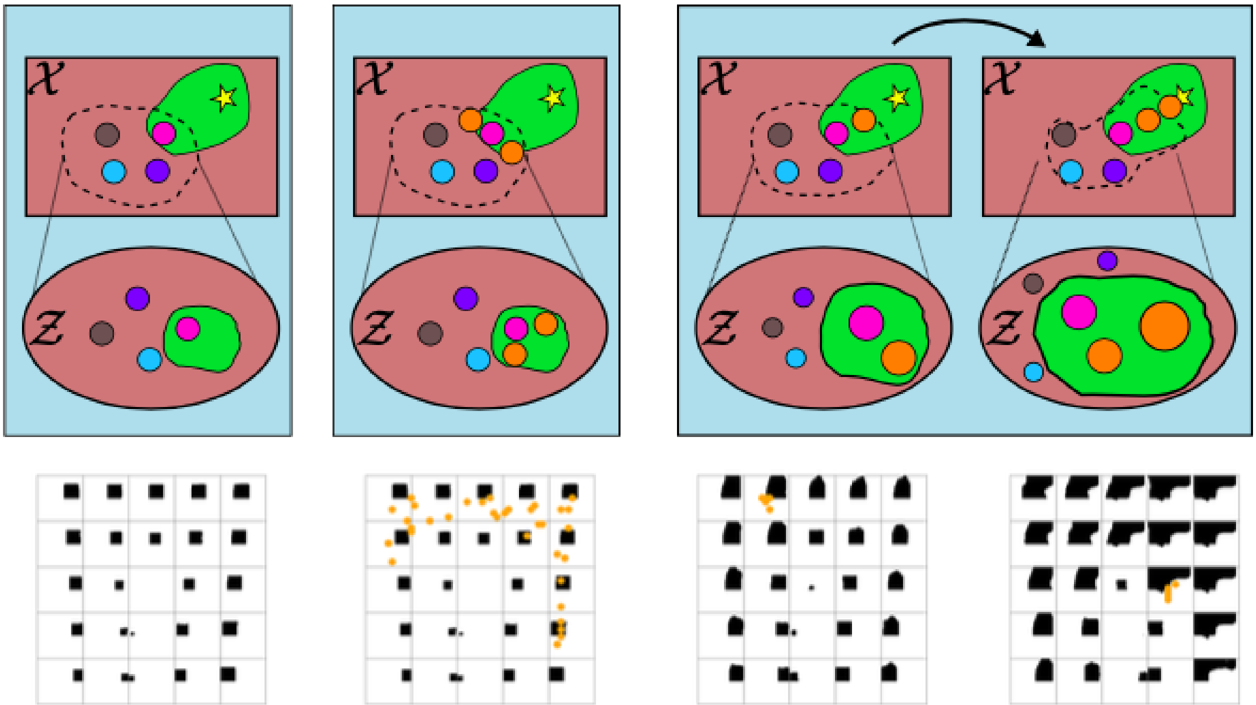
- 使用一个obective/surrogate model $h_{\mathcal{X}} : \mathcal{X} \mapsto \mathbb{R}$ 来近似 f .优化 h 要比 f 容易的多。 $f(\mathbf{x}) \approx h(\mathbf{x})$
- 但由于 $\mathcal{X}$ 空间的复杂，使得优化 $h_{\mathcal{X}}$ 也变得困难
- $h_{\mathcal{X}}$ 需要用样本{ $(\mathbf{x}, f(x)$ }来拟合

Latant Space Optimization（LSO）

- 建立一个生成模型： $g : \mathcal{Z} \mapsto \mathcal{X}$.（可以使用VAE、GAN）
- 使用一个**latent** obective/surrogate model $h : \mathcal{Z} \mapsto \mathbb{R}$ 来近似 f . $f(g(\mathbf{z})) \approx h(\mathbf{z})$
- 如果 $\mathcal{Z}$ 选择低维、连续空间，如 $\mathbb{R}^n$ ，优化 $h_{\mathcal{Z}}$ 就变简单了
- $h_{\mathcal{Z}}$ 需要用样本{ $(\mathbf{z}, f(g(\mathbf{z}))$ }来拟合，所以需要一个解码器 $q : \mathcal{X} \mapsto \mathcal{Z}$

LSO NOT working well

- 首先生成模型有个特点：生成模型 $g : \mathcal{Z} \mapsto \mathcal{X}$ ，SOTA的模型如VAE、GAN训练中需要一个prior $p(\mathbf{z})$
  - 这意味着尽管 g 是定义在整个 $\mathcal{Z}$ 上的，但几乎全部概率集中在一个子空间内 $\mathcal{Z}' \subset \mathcal{Z}$ ， $\mathcal{Z}'$ 称为隐空间的可行域。在这个可行域以外的sample一般结果很差或者invalid structure
  - 所以一般都会在可行域 $\mathcal{Z}'$ 内进行优化。*bounded* optimization
- 对很多优化问题，DGM训练数据大部分是low-score(sub optimal)=>致使可行域大部分都是low-score点（使求解问题变得“大海捞针”）、把稀有的ligh-score的点放到DGM training distribution外



(a) Starting Point (b) Standard LSO (c) LSO with Weighted Retraining

Figure 1: Schematic illustrating LSO with and without weighted retraining. The cartoon illustrates the input/latent space of the generative model (**top**). The latent manifold from Section 6.2's 2D shape area maximization task is shown for comparison (**bottom**). Each image in the manifold shows the result of decoding a latent point on a uniform square grid in a 2D latent space; images are centered on the original grid points. Red/green regions correspond to points with low/high objective function values respectively. The yellow star is the global optimum in $\mathcal{X}$. Coloured circles are data points; their radius represents their weight. The dashed line surrounds the region of $\mathcal{X}$ modelled by g (i.e. $g(\mathcal{Z})$, the image of $\mathcal{Z}$). **(a)** The status of the generative model g at the start of optimization. **(b)** The result of standard LSO with g fixed, which queries the points in orange. It is only able to find points close to the training data used to learn $\mathcal{Z}$, resulting in slow and incomplete exploration of $\mathcal{X}$. **(c)** The result midway (left) and at the end (right) of LSO with our proposed approach, which weights data points according to their objective function value and retrains g to incorporate newly queried data. This continually adjusts $\mathcal{Z}$ to focus on modelling the most promising regions of $\mathcal{X}$, speeding up the optimization and allowing for substantial extrapolation beyond the initial training data.

- 生成模型的目标和实际的目标不一致
  - 生成模型：为了学习latent space能尽可能接近训练数据的分布
  - 实际的目标：为了学习latent space能易于优化算法在latent space上高效优化
- 随着新的数据点的加入，没有shift latent space。整个过程固定了 g

LSO with Weighted Retraining

- weighting
  - 简单的在DGM训练中丢掉low-score 点仅仅在数据点多的时候有效(X)
  - 让high-score在latent space上有更高的概率，low-score在latent space上更低的概率。使用所有data训练学习representation防止过拟合
    - 具体做法1：为每个data points赋予一个大于等于0的权重。score越高，权重越大。所有数据权重求和为1
    - 具体做法2: 权重作为采样概率来构成minibatch训练

- 一种rank可选方式，k是超参：

$$w(\mathbf{x}; \mathcal{D}, k) \propto \frac{1}{kN + \mathrm{rank}_{f,\mathcal{D}}(\mathbf{x})}, \quad \mathrm{rank}_{f,\mathcal{D}}(\mathbf{x}) = |\{\mathbf{x}_i : f(\mathbf{x}_i) > f(\mathbf{x}),\ \mathbf{x}_i \in \mathcal{D}\}| \ , \tag{1}$$

- Retraining
  - fine-tuning
  - 若新的data带有high-score，那么会赋予大的权重同时给旧data小的权重。对training distribution影响较大。因此latent space会较大程度改变
  - 若新的data带有low-score，那么会赋予小的权重，同时旧data权重改变不大。基本不改变training distribution。因此latent space不怎么变化

A.3.1 PyTorch (weighted sampling)

Standard Training

```
from torch.utils.data import *


dataloader = DataLoader(data)
for batch in dataloader:
    # ...
```

Weighted Training

```
from torch.utils.data import *
sampler = WeightedRandomSampler(
    weights, len(data))
dataloader = DataLoader(data, sampler=sampler)
for batch in dataloader:
    # ...
```

A.3.2 PyTorch (direct application of weights)

Standard Training

```
criterion = nn.MSELoss()
outputs = model(inputs)
loss = criterion(outputs, targets)

loss.backward()
```

Weighted Training

```
criterion = nn.MSELoss(reduction=None)
outputs = model(inputs)
loss = criterion(outputs, targets)
loss = torch.mean(loss * weights)
loss.backward()
```

Algorithm 1 Latent Space Optimization with Weighted Retraining (changes highlighted in blue)

1: **Input:** Data $\mathcal{D} = \{(\mathbf{x}_i, f(\mathbf{x}_i))\}_{i=1}^N$, query budget M, objective function $f(\mathbf{x})$, latent objective model $h(\mathbf{z})$, generative/inverse model $g(\mathbf{z})/q(\mathbf{x})$, retrain frequency r, weighting function $w(\mathbf{x})$
2: **for** $1, \ldots, M/r$ **do**
3: Train generative model g and inverse model q on data $\mathcal{D}$ weighted by $\mathcal{W} = \{w(\mathbf{x})\}_{\mathbf{x} \in \mathcal{D}}$
4: **for** $1, \ldots, r$ **do**
5: Compute latent variables $\mathcal{Z} = \{\mathbf{z} = q(\mathbf{x})\}_{\mathbf{x} \in \mathcal{D}}$
6: Fit objective model h to $\mathcal{Z}$ and $\mathcal{D}$, and optimize h to obtain new latent query point $\tilde{\mathbf{z}}$
7: Obtain corresponding input $\tilde{\mathbf{x}} = g(\tilde{\mathbf{z}})$, evaluate $f(\tilde{\mathbf{x}})$ and set $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\tilde{\mathbf{x}}, f(\tilde{\mathbf{x}}))\}$
8: **end for**
9: **end for**
10: **Output:** Augmented dataset $\mathcal{D}$

实验

Results

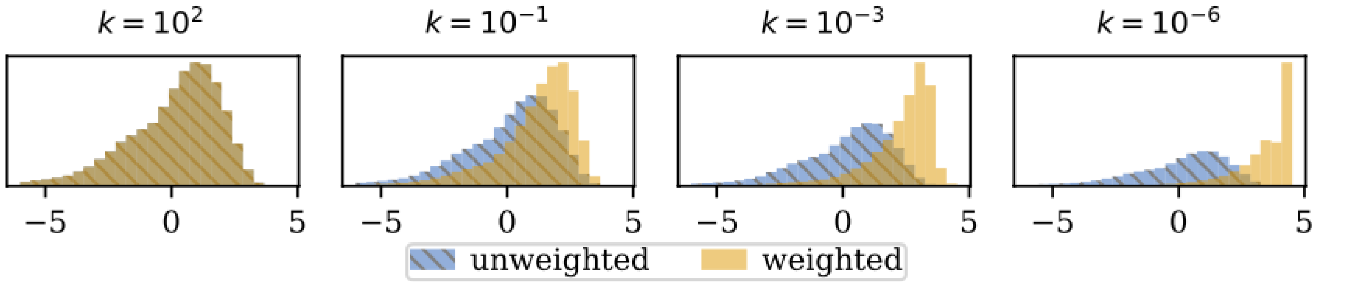


Figure 2: Histogram of objective function values for the ZINC dataset (see Section 6) with uniform weighting (in blue) as well as rank weighting from Equation (1) for different k values (in orange). Large k approaches uniform weighting, while small k places most weight on high-scoring points.

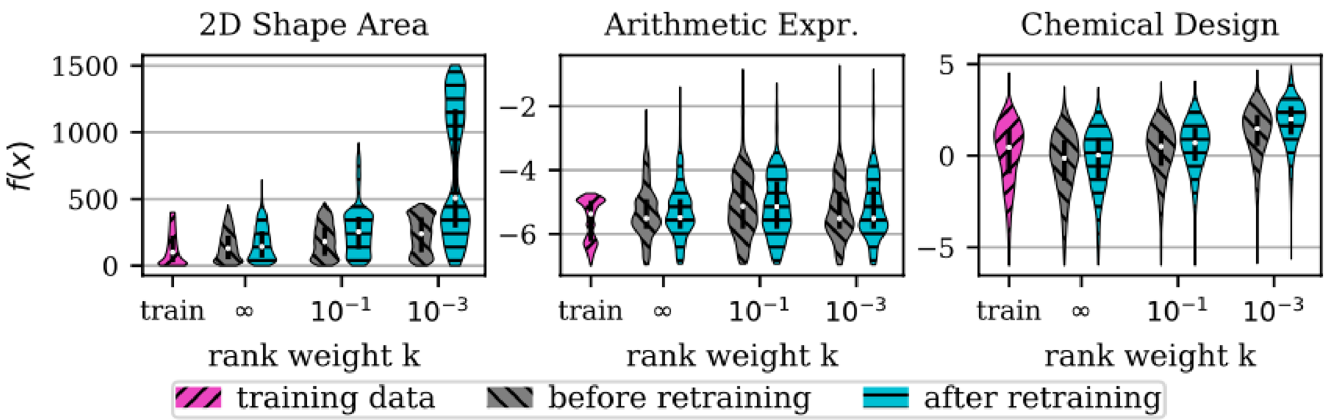


Figure 3: Objective value distribution for the training set and samples from the DGM's prior for all three tasks for different k values, before and after weighted retraining (see Section 6.2).

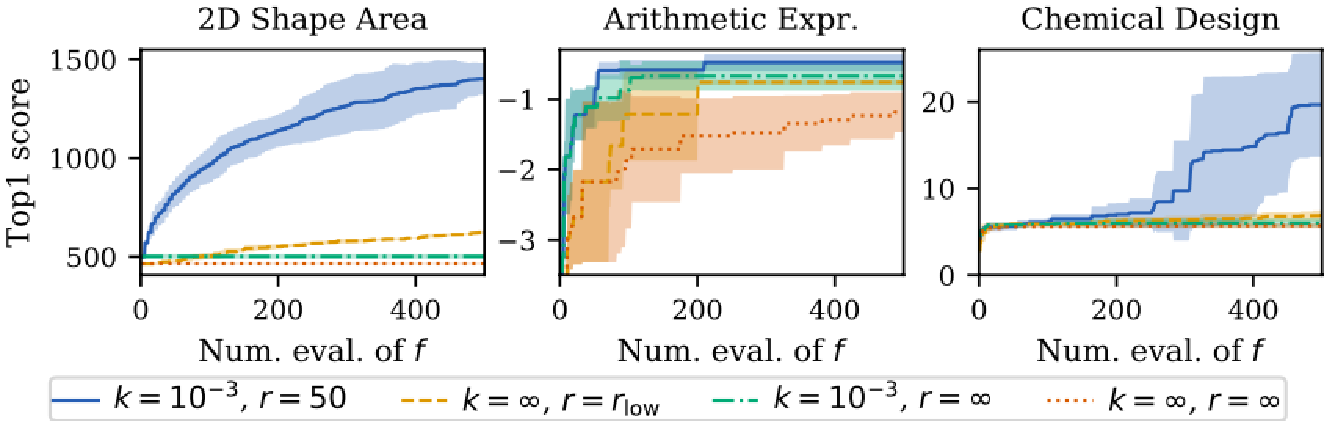


Figure 4: Top1 optimization performance of weighted retraining for all tasks, for different k values (i.e. $k \in \{10^{-3}, \infty\}$) and retraining frequencies (i.e. $r_{\text{low}} = 5$ for the 2D shape area task, and $r_{\text{low}} = 50$ for the other two tasks). Shaded area corresponds to standard deviation.

- r是retraining 周期

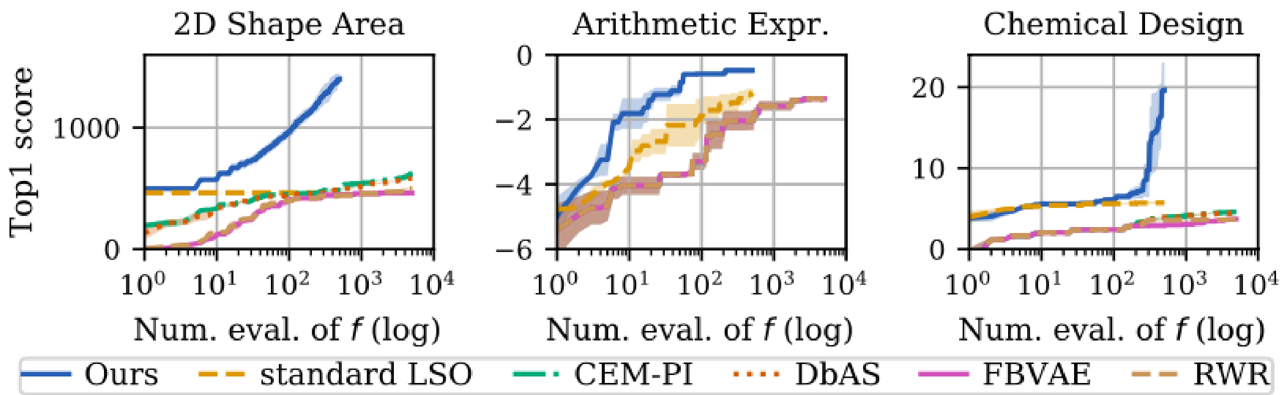


Figure 5: Comparison of weighted retraining, LSO, CEM-PI, DbAS, FBVAE and RWR. Our approach significantly outperforms all baselines, achieving both better sample-efficiency and final performance.

Independence from Dataset Size

$$
\begin{aligned}
\sum_{r=q_1 N}^{q_2 N} w(\mathbf{x}_r; \mathcal{D}, k) &= \sum_{r=q_1 N}^{q_2 N} \frac{1}{kN + r} \\
&= \sum_{r=1}^{(k+q_2)N} \frac{1}{r} - \sum_{r'=1}^{(k+q_1)N-1} \frac{1}{r'} \\
&\approx \left(\ln\left((k+q_2)N - 1\right) + \gamma\right) - \left(\ln\left((k+q_1)N - 1\right) + \gamma\right) \\
&= \ln \frac{(k+q_2)N}{(k+q_1)N - 1} \approx \ln \frac{(k+q_2)N}{(k+q_1)N} = \ln \frac{(k+q_2)}{(k+q_1)}
\end{aligned}
$$

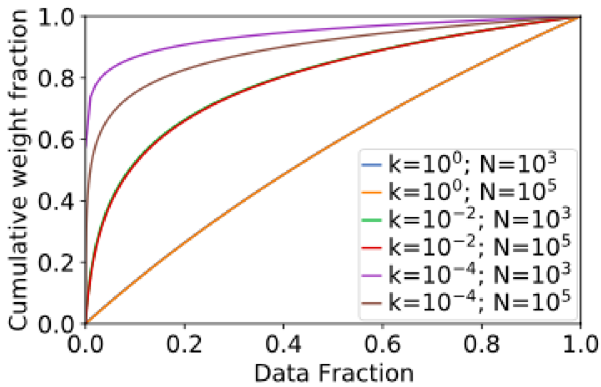


Figure 6: Cumulative distribution of rank weights (sorted highest to lowest), showing a distribution that is independent of N if $kN > 1$.