真的是作业

# 深度生成模型中隐空间局部最优方向推荐问题

梁天一[1*]，陈昌谷[2*]

1. 华东师范大学计算机科学与技术, 上海
2. 华东师范大学计算机科学与技术, 上海
学号: 51215901019, 52215901006

## 1 问题引入

用深度生成模型生成各种模态数据的方法发展迅猛, 比如生成图像 [1, 3, 3] 和声音 [4], 乃至 3D 模型 [5]. 这些方法成功的关键是能够使用具有更多层和更高维度内部表示 (即隐空间) 的复杂网络架构。比如 3D 生成模型 IMAE [5] 有 128 维的隐空间. 这种网络架构的丰富性导致了内部隐变量和真实数据之间高度非线性的表示，从而能够生成几乎与真实数据难以区分的高质量数据。当基础生成网络架构发展逐渐成熟，可以直接生成大量高保真且多样的高质量生成物后，问题的关键转化为如何从训练完成的生成模型（预训练模型）中，挖掘出我们想要的生成物。这种挖掘被表述为在生成模型的隐空间中，找到一组隐向量，然后通过生成模型把隐向量映射为生成物，并检验这些生成物是否满足用户偏好。这类挖掘预训练生成模型的工作非常好，简直就是发论文的源泉。别人用它做了图像声音，我转手就倒卖到几何建模领域，而且我每个几何模型靠美工都要花费相当长的时间来完成，而且美工要熟练掌握犀牛，3Dmax 等专业软件。我这边只要准备好数据集 train 一发，之后部署这个来挑选一波，就算每个模型打 1 折都可以挂到模型网站血赚一波。回头搞个孵化，融资的时候吹吹元宇宙，区块链中的 NFT，财务自由美滋滋。(拉回来拉回来)

完成这一挖掘是极其富有挑战性的。一方面是因为**用户偏好是因人而异，随时间有扰动，因此难以用封闭形式的数学模型来刻画**。所以需要引入用户参与进行探索。现有方法主要用在描述子提取的特征空间中，尝试给出一组交互策略，让用户通过尽可能地少量交互，就可以搜寻到用户满意的物体。和生成模型的挖掘任务需要创造样本不同，这种方法仅仅是一种搜索或者溯源任务，因为所以样本都来自现实中已经存在的数据集。生成模型隐空间中除了已经存在的数据集的投影，还可以在这些可列个点的不可数补集中找到大量生成物对应的隐向量。另一方面，是因为隐空间**高维且难以估计数据先验分布**。隐向量十分高维，这意味贝叶斯优化难以胜任。且对于一般自编码器中隐空间数据作为先验分布而言是多峰难以估计的。通过推荐一些隐空间的方向，组成滑条，用户可以通过滑动滑条来改变隐向量，进而得到了搜索空间。然而, 在这种几百维的高维空间找到合适的修改操作是十分困难的。最近针对预训练的生成模型，无监督的挖掘可解释的方向 [8] 为这类方法的解决提供了希望。

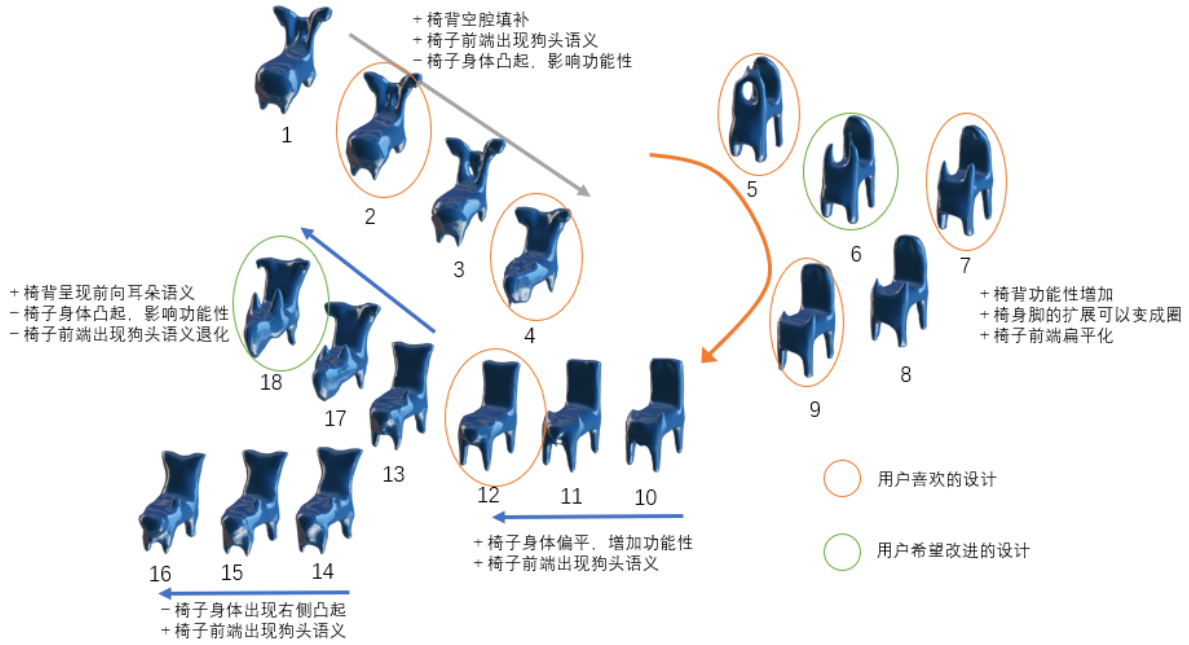


**图 1** 隐空间探索的案例：用户从一个初始生成物出发，通过被推荐的 TopK 个最优方向所构成的滑条来移动隐向量（矢量表示），然后输入新的隐向量到生成器得到新的生成物，不同序号表示通过一次滑动得到的新的生成物。由于没有样本标签，被推荐的方向的语义通常由用户进行后解释。

## 2 问题定义

我们要解决的问题是根据给定**预训练的生成模型**以及初始生成物对应的隐变量，允许用户用**尽可能少的交互次数**来找到和初始生成物相似的，让用户满意的**变体**.

符号定义

生成模型 $f : \mathbb{Z} \to \mathbb{X}$ 其中 $z \in \mathbb{Z}$ 是隐变量，一般 $z$ 是几百维的向量，$x \in \mathbb{X}$ 是生成物，比如在图像中是 $R^{HWC}$ 的张量。用户仅能够操控 $\mathbb{Z}$ 中向量来利用模型 $f$ 生成 $x$。用一个偏好函数来刻画用户满意度 $g : \mathbb{X} \to \mathbb{R}$ 用户对这个结果 $x \in \mathbb{X}$ 越满意，意味着生成物的质量分数越高. 这个偏好函数是未知的，而且因人而异，甚至在探索任务过程中会出现模糊或变化。

于是我们要求解以下优化问题：

$$z^{\circ} = \arg\max_{z \in Z} g(f(z)) \tag{1}$$

因为 $g$ 是为未知的，我们要迭代更新当前最优隐向量 $z^{(k)}$：

$$z^{(k+1)} = z^{(k)} + \Delta^{(k)} = z^{(k)} + \alpha(\frac{\partial g(z^{(k)})}{\partial z})^T, \tag{2}$$

其中 $k$ 是迭代数，$\Delta^{(k)} = \alpha(\frac{\partial g(z^{(k)})}{\partial z})^T$ 就是**推荐方向向量**，$\alpha > 0$ 是迭代的步长，那么这个步长最优情况为让偏好函数提升最高的值：

$$\alpha = \arg\max_{\dot{\alpha}} g(f(z^{(k)} + \hat{\alpha}(\frac{\partial g}{\partial z})^T)) \tag{3}$$

## 3 优化方法

我们的方法是建立在优化的概念上, 是为了探索目的而设计的。由于隐向量与生成数据之间的关系通常是高度非线性且多峰值的, 因此根据优化的一般理解很难找到准确的目标 (即全局最优解) [9]。尽管如此，找到局部最优值的能力仍然对用户调整结果和探索相邻选项有用。

我们的关键思想是让用户执行迭代搜索，每个迭代搜索都在一个大大降维的子空间中。这样的子空间是在生成模型的局部微分分析的基础上构造的。为了进行高效的搜索，子空间中进行一个微小改变需要对结果数据产生足够的改变。说到**子空间**和**足够**，很自然地，显然，答案略地，容易想到可以用线性变换的特征子空间和特征向量来刻画这件事情，然后今年下半年就可以用中美合拍的深度生成模型随意找到自己满意的生成物，实现文体两开花。因此方法就是在生成模型的雅可比矩阵上进行奇异值分解, 提取一个低维子空间。根据奇异值随机选择一组奇异向量，形成一个子空间，以保留访问所有子空间的机会 (例如, 遍历性)。在交互的时候，通过 1D 滑块界面向用户呈现 1D 子空间。从初始位置开始，用户在呈现的 1D 子空间中找到一个新的候选空间，然后在新的候选位置更新该子空间。此过程将重复进行，直到无法进一步改进为止。

我们把低维子空间中的向量定义为 $w \in \mathcal{W}(z^{(k)}) \subset \mathbb{R}^l$，其中 $l \ll n$。然后带入到公式2中:

$$\begin{aligned} z^{(k+1)} &= z^{(k)} + \tilde{\Delta}^{(k)} = z^{(k)} + p_{z^{(k)}(w)}, \\ w &= \underset{\hat{w} \in W(z^{(k)})}{\arg\max}\, g(f(z^{(k)} + p_{z^{(k)}}(\hat{w}))), \end{aligned} \tag{4}$$

其中 $p_{z^{(k)}} : \mathcal{W}(z^{(k)}) \to \mathcal{Z}$ 是一个算子, 把低维向量 $w$ 和当前隐向量 $z^{(k)}$ 投射到原空间 $\mathcal{Z}$。为了用户交互方便, $l = 1$, 这样就可以当做滑条用了。接下来要考虑 1 如何选择合适的子空间 $\mathcal{W}(z^{(k)})$;2 如何保证 $\mathcal{Z}$ 空间中所有候选的生成物都可以被等概率地保留。

### 3.1 子空间构建.avi

首先考虑用链式法则展开 $\frac{\partial g}{\partial z}$:

$$(\frac{\partial g}{\partial z})^T = (\frac{\partial f}{\partial z})^T (\frac{\partial g}{\partial f})^T \tag{5}$$

雅克比矩阵为:

$$J = \frac{\partial f}{\partial z} = \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial z_1} & \cdots & \frac{\partial f_m}{\partial z_n} \end{bmatrix} \in R^{m \times n}, \tag{6}$$

对雅克比矩阵做奇异值分解:

$$J = U \Sigma V^T \tag{7}$$

所谓低维子空间，其实就是用一个 $l$ 维的 $\tilde{\Sigma}$ 来近似 $\Sigma$。关于如何等概率地保留所有候选的生成物，其实就是用期望相等来刻画，即 $E[\tilde{\Sigma}] = \Sigma$。这种无偏性可以用概率与奇异值的大小成正比的随机采样来完成。这样改写一下公式7就可以得到:

$$J \approx \tilde{J} = U \tilde{\Sigma} V^T \tag{8}$$

那么公式2中的推荐方向 $\Delta^{(k)}$ 既可以被近似为:

$$\begin{aligned}\Delta^{(k)} &= \alpha(\frac{\partial g(z^{(k)})}{\partial z})^T = \alpha J^T(\frac{\partial g}{\partial f})^T \\ &\approx \tilde{\Delta}^{(k)} = \alpha J^T(\frac{\partial g}{\partial f})^T = \alpha \hat{V}\dot{\Sigma}^T\hat{U}^T(\frac{\partial g}{\partial f})^T \\ &= \hat{V}w = p_{z^{(k)}}(w),\end{aligned} \tag{9}$$

其中 $w = \alpha\hat{\Sigma}^T\hat{U}^T(\frac{\partial g}{\partial f})^T$ 是 $l$ 维向量，在被选中的列向量 $\hat{V}$ 组成的子空间 $\mathcal{W}(z^{(k)})$。这样一来 $E[\tilde{\Delta^{(k)}}] = \Delta^{(k)}$。

### 3.2 搜索.jpg

如算法1所示，用户首先滑动滑条，用户停下的地方就是用户偏好函数 $g$ 最大的地方，一旦用户找到一个新的隐向量 $z^{(k+1)} = z^{(k)} + \tilde{\Delta}^{(k)}$，用户停下来点击更新推荐方向。这时候因为 $z$ 改变了，雅克比矩阵 $J$ 也变了，自然推荐的方向 $w = \alpha\hat{\Sigma}^T\hat{U}^T(\frac{\partial g}{\partial f})^T$ 也变了。用户接着使用更新后的方向构成的滑条继续探索，直到他/她/它/🄴满意为止。

**算法 1** 可微子空间搜索

**输入:** 用户偏好函数 $g$ 生成模型 $f$ 初始隐向量 $z$;

**主迭代:**

$k \leftarrow 0; z^* \leftarrow NULL$
**while** User not satisfied with $f(z^{(k)})$ **do**
  $\mathcal{W}(z^{(k)}), \rho_{z^{(k)}} \leftarrow SVD_Reduction(\ \partial f_{\overline{\partial z}})$;
  User finds $w^{(k)} \in \mathcal{W}(z^{(k)})$;
  $z^{(k+1)} \leftarrow z^{(k)} + p_{z^{(k)}}(w^{(k)})$;
  $k \leftarrow k + 1$;
  $z^* \leftarrow z^{(k+1)}$;
**end while**

**输出:** $z^*$

## 4 工程实现

拿出祖传的 IMAE 自编码器，开始操作隐向量。

代码链接：https://gist.github.com/tianyilt/bc2045209a27d7ecc04eefa6790e8e37

**Listing 1** Sample Python code – Local Optimal Direction Recommendation in Latent Space of Deep Generative Model.

```
def init_iteration(self, new_p):
    self.current_data = self.model.decode(new_p.reshape(1, -1))[0]
    if self.update_as_best or self.search_iter_count == 0:
        self.current_best_data = self.current_data

    self.system_points = []
    self.system_points.append(new_p)
    self.user_points = []
```

```
        self.user_points.append(np.zeros(self.ui_space_size))

        self.update_subspace(new_p)

        self.current_updated_flag = True
        self.target_updated_flag = True
        self.current_best_updated_flag = True

    def update_subspace(self, p):
        self.jacobian = self.model.calc_model_gradient(p.reshape(1, self.latent_size))
        u, s, vh = np.linalg.svd(self.jacobian, full_matrices=True)

        num = s.shape[0]

        self.jacobian_vhs = vh[:num]
        self.jacobian_s = s[:num]
        self.jacobian_mask = np.ones(self.jacobian_vhs.shape[0], dtype=bool)

        self.importance_sampling_subspace_basis()
        print('end updating subspace!')

    def importance_sampling_subspace_basis(self):
        if self.jacobian_s[self.jacobian_mask].shape[0] < self.subspace_dim:
            self.jacobian_mask = np.ones(self.jacobian_vhs.shape[0], dtype=bool)

        s = self.jacobian_s[self.jacobian_mask] + 1e-6
        vh = self.jacobian_vhs[self.jacobian_mask]

        # Importance sampling
        choice_p = s / np.sum(s)
        idx = np.random.choice(choice_p.shape[0], self.subspace_dim, replace=False, p=↩
            choice_p)

        self.subspace_basis = vh[idx]

        self.jacobian_mask[np.arange(self.jacobian_vhs.shape[0])[self.jacobian_mask][↩
            idx]] = False


    def calc_model_gradient(self, latent_vector):
        if self.use_approx:
            idx = np.random.choice(self.time_per_axis3 * 32768, 50, replace=False)
        else:
            idx = np.arange(self.time_per_axis3 * 32768).reshape(-1, 1)
```

```
51
52      counter = 0
53      gradient = np.zeros((idx.shape[0], self.z_dim))
54      for i in range(self.time_per_axis3):
55          mask = (idx >= i * 32768) & (idx < (i + 1) * 32768)
56          tmp_idx = idx[mask] - i * 32768
57          tmp_gradient = self.sess.run(self.gradient, feed_dict={self.z: ↩
                latent_vector, self.point_coord: self.coords[i], self.random_idx: ↩
                tmp_idx.reshape(-1, 1)}).reshape(-1, self.latent_size)
58          gradient[counter:counter + tmp_gradient.shape[0], :] = tmp_gradient
59          counter += tmp_gradient.shape[0]
60      return gradient
61
62  def build(self):
63      with self.graph.as_default():
64          self.zGAN_output = self.buildZGAN()
65          self.buildIMAE()
66
67          self.random_idx = tf.placeholder(tf.int32, shape=(None, 1), name='↩
                random_idx')
68          with tf.name_scope('Gradient'):
69              slices = tf.gather(tf.reshape(self.zD, [1, -1]), self.random_idx[..., ↩
                    0], axis=1)
70              self.gradient = self.jacobian(slices, self.z, parallel_iterations=1)
71
72
73  def jacobian(self, fx, x, parallel_iterations=10):
74      dtype = x.dtype
75
76      n = tf.shape(fx)[1]
77      loop_vars = [
78          tf.constant(0),
79          tf.TensorArray(dtype, size=n),
80      ]
81
82      _, grads = tf.while_loop(
83          lambda j, _: j < n,
84          lambda j, result: (j + 1, result.write(j, tf.gradients(tf.gather(fx, j, ↩
                axis=1), x))),
85          loop_vars,
86          parallel_iterations=parallel_iterations
87      )
88      grads = grads.stack()
89      grads = tf.transpose(grads, [2, 1, 0, 3])
```

```
90      grads = tf.reshape(grads, [tf.shape(fx)[0], tf.shape(fx)[1], tf.shape(x)[1]])
91
92      return grads
93
94
95  from .GlobalOptimizer import GlobalOptimizer
96
97  import numpy as np
98
99  class JacobianOptimizer(GlobalOptimizer):
100     def __init__(self, n, m, f, g, search_range, jacobian_func, maximizer=True):
101         super(JacobianOptimizer, self).__init__(n, m, f, g, search_range, ↩
                maximizer=maximizer)
102         self.name = 'Jacobian Optimizer'
103         self.jacobian_func = jacobian_func
104         self.center_tolerance = 0.05
105
106     def init(self, init_z):
107         super(JacobianOptimizer, self).init(init_z)
108
109         self.update_jacobian(self.current_z)
110         self.sample_direction()
111
112     def get_z(self, t):
113         return self.current_z + self.subspace_basis * (t - 0.5) * self.↩
                search_range * 2
114
115     def update_jacobian(self, z):
116         self.jacobian = self.jacobian_func(z)
117         u, s, vh = np.linalg.svd(self.jacobian, full_matrices=True)
118
119         num = s.shape[0]
120         self.jacobian_vhs = vh[:num]
121         self.jacobian_s = s[:num]
122         self.jacobian_mask = np.ones(self.jacobian_vhs.shape[0], dtype=bool)
123
124     def sample_direction(self):
125         if self.jacobian_s[self.jacobian_mask].shape[0] <= 0:
126             self.jacobian_mask = np.ones(self.jacobian_vhs.shape[0], dtype=bool)
127
128         s = self.jacobian_s[self.jacobian_mask] + 1e-6
129         vh = self.jacobian_vhs[self.jacobian_mask]
130
131         # Importance sampling
```

```
132         choice_p = s / np.sum(s)
133         idx = np.random.choice(choice_p.shape[0], p=choice_p)
134 
135         self.subspace_basis = vh[idx]
136 
137         self.jacobian_mask[np.arange(self.jacobian_vhs.shape[0])[self.↩
                jacobian_mask][idx]] = False
138 
139     def update(self, t):
140         self.current_z = self.get_z(t)
141         self.current_x = self.f(self.current_z.reshape(1, -1))[0]
142         self.current_score = self.g(self.current_x.reshape(1, -1))[0]
143 
144         if np.abs(t - 0.5) > self.center_tolerance:
145             self.update_jacobian(self.current_z)
146         self.sample_direction()
147 
148 
149 class GlobalOptimizer():
150     def __init__(self, n, m, f, g, search_range, maximizer=True):
151         self.n = n
152         self.m = m
153         self.f = f
154         self.g = g
155         self.search_range = search_range
156         self.maximizer = maximizer
157 
158     def init(self, init_z):
159         self.current_z = init_z
160         self.current_x = self.f(self.current_z.reshape(1, -1))[0]
161         self.current_score = self.g(self.current_x.reshape(1, -1))[0]
162 
163         print('Initialize', self.name, 'with score', self.current_score )
164 
165     def get_z(self, t):
166         pass
167 
168     def find_optimal(self, sample_n, batch_size=100):
169         if batch_size <= 0:
170             batch_size = sample_n
171 
172         zs = []
173         for i in range(sample_n):
174             t = i / float(sample_n - 1)
```

```
            z = self.get_z(t)
            zs.append(z)
        zs = np.array(zs)

        batch_n = sample_n // batch_size
        remainder = sample_n - batch_size * batch_n

        xs = np.zeros((sample_n, self.m))
        for i in range(batch_n):
            xs[i * batch_size:(i + 1) * batch_size] = self.f(zs[i * batch_size:(i + 1) * batch_size])
        if remainder != 0:
            xs[batch_n * batch_size:] = self.f(zs[batch_n * batch_size:])

        scores = self.g(xs)

        if self.maximizer:
            idx = np.argmax(scores)
        else:
            idx = np.argmin(scores)
        t = idx / float(sample_n - 1)

        z = self.get_z(t)
        x = self.f(z.reshape(1, -1))[0]
        score = self.g(x.reshape(1, -1))[0]

        return z, x, score, t

    def update(self, t):
        pass
```

结果如1所示。

## 参考文献


1 Karras T, Aila T, Laine S, et al. Progressive growing of gans for improved quality, stability, and variation[J]. arXiv preprint arXiv:1710.10196, 2017.

2 Kingma D P, Mohamed S, Rezende D J, et al. Semi-supervised learning with deep generative models[C]//Advances in neural information processing systems. 2014: 3581-3589.

3 Kingma D P, Mohamed S, Rezende D J, et al. Semi-supervised learning with deep generative models[C]//Advances in neural information processing systems. 2014: 3581-3589.

4 Engel J, Agrawal K K, Chen S, et al. GANSynth: Adversarial Neural Audio Synthesis[C]//International Conference on Learning Representations. 2018.

5 Chen Z, Zhang H. Learning implicit fields for generative shape modeling[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 5939-5948.

6 Author A, Author B, Author C. Reference title. Journal, Year, Vol: Number or pages

7 张三, 李四, Author C, et al. Reference title. In: Proceedings of Conference, Place, Year. Number or pages

8 Härkönen E, Hertzman A, Lehtinen J, et al. GANSpace: Discovering Interpretable GAN controls[C]//IEEE Conference on Neural Information Processing Systems;. 2020.

9 Nesterov Y. Lectures on convex optimization[M]. Berlin, Germany: Springer International Publishing, 2018.

10 Chiu C H, Koyama Y, Lai Y C, et al. Human-in-the-loop differential subspace search in high-dimensional latent space[J]. ACM Transactions on Graphics (TOG), 2020, 39(4): 85: 1-85: 15.

本研究几乎处处基于文献 [10] 改编 a.e.。