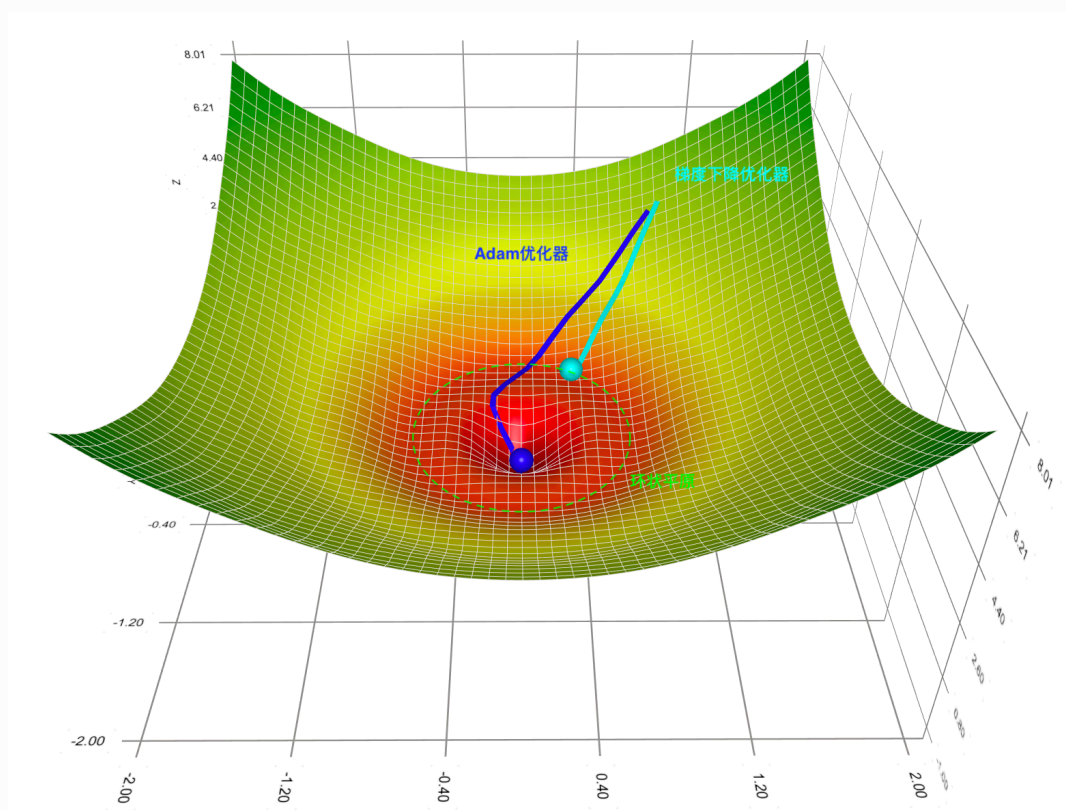


量子神经网络的贫瘠高原效应 (BARREN PLATEAUS)

Copyright (c) 2020 Institute for Quantum Computing, Baidu Inc. All Rights Reserved.

概览

在经典神经网络的训练中，基于梯度的优化方法不仅仅会遇到局部最小值的问题，同时还要面对鞍点等附近梯度近似于零的几何结构。相对应的，在量子神经网络中也存在着一种**贫瘠高原效应** (Barren plateaus)。这个奇特的现象首先是由 McClean et al. 在 2018 年发现 ([arXiv:1803.11173](https://arxiv.org/abs/1803.11173))。简单来说，就是当你选取的随机电路结构满足一定复杂程度时优化曲面 (Optimization landscape) 会变得很平坦，从而导致基于梯度下降的优化方法很难找到全局最小值。对于大多数的变分量子算法 (VQE 等等)，这个现象意味着当量子比特数量越来越多时，选取随机结构的电路有可能效果并不好。这会让你设计的损失函数所对应的优化曲面变成一个巨大的高原，让从而导致量子神经网络的训练愈加困难。你随机找到的初始值很难逃离这个高原，梯度下降收敛速度会很缓慢。



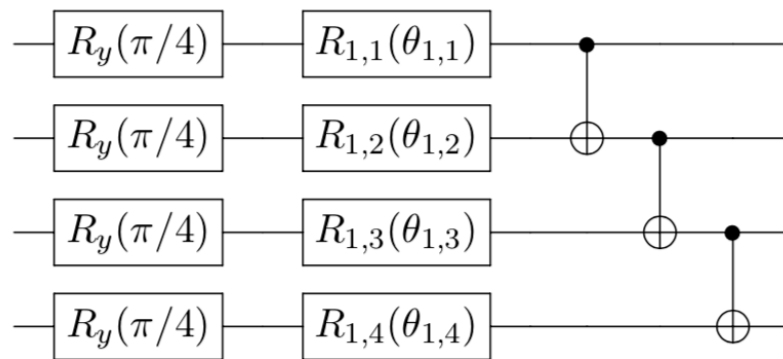
图片由 Gradient Descent Viz 生成

本教程主要讨论如何在量桨（PaddleQuantum）平台上展示贫瘠高原这一现象。其中并不涉及任何算法创新，但能提升读者对于量子神经网络训练中梯度问题的认识。首先我们先引入必要的 library 和 package：

```
1 import time
2 import numpy as np
3 from matplotlib import pyplot as plt
4 import paddle.fluid as fluid
5 from paddle.fluid.framework import ComplexVariable
6 from paddle.complex import matmul, transpose
7 from paddle_quantum.circuit import UAnsatz
8 from paddle_quantum.utils import hermitian
9 from paddle_quantum.state import density_op
```

随机的网络结构

这里我们按照原作者 McClean (2018) 论文中提及的类似方法搭建如下随机电路（目前我们不支持内置的 control-Z 门，方便起见用 CNOT 替代）：



首先作用在所有量子比特上绕布洛赫球的 Y-轴旋转 $\pi/4$ 。

其余的结构加起来构成一个模块（Block），每个模块共分为两层：

- 第一层搭建随机的旋转门，其中 $R_{\ell,n} \in \{R_x, R_y, R_z\}$ 。下标 ℓ 表示处于第 ℓ 个重复的模块，上图中 $\ell = 1$ 。第二个下标 n 表示作用在第几个量子比特上。
- 第二层由 CNOT 门组成，作用在每两个相邻的量子比特上。

在量桨中，我们可以这么搭建。

```

1  def rand_circuit(theta, target, num_qubits):
2      # 我们需要将 Numpy array 转换成 Paddle 动态图模式中支持的 variable
3      const = fluid.dygraph.to_variable(np.array([np.pi/4]))
4
5      # 初始化量子电路
6      cir = UAnsatz(num_qubits)
7
8      # ===== 第一层 =====
9      # 固定角度的 Ry 旋转门
10     for i in range(num_qubits):
11         cir.ry(const, i)
12
13     # ===== 第二层 =====
14     # target是一个随机的数组,用来帮助我们抽取随机的单比特门
15     for i in range(num_qubits):
16         if target[i] == 0:
17             cir.rz(theta[i], i)
18         elif target[i] == 1:
19             cir.ry(theta[i], i)
20         else:
21             cir.rx(theta[i], i)
22
23     # ===== 第三层 =====
24     # 搭建两两相邻的 CNOT 门
25     for i in range(num_qubits - 1):
26         cir.cnot([i, i + 1])
27
28     return cir.U

```

损失函数与优化曲面

当我们确定了电路的结构之后，我们还需要定义一个损失函数（Loss function）来确定优化曲面（Optimization landscape）。按照原作者 McClean (2018)论文中提及的，我们采用 VQE算法中常用的损失函数：

$$\mathcal{L} = \langle 0|U^\dagger(\boldsymbol{\theta})HU(\boldsymbol{\theta})|0\rangle$$

其中的酉矩阵 $U(\boldsymbol{\theta})$ 就是我们上一部分搭建的带有随机结构的量子神经网络。对于其中的哈密顿量 H 我们不妨先取最简单的形式 $H = |00\cdots 0\rangle\langle 00\cdots 0|$ 。设定好这些后，我们就可以从最简单的两个量子比特的情形开始采样了 -- 生成 300 组随机网络结构和不同的随机初始参数 $\{\theta_{\ell,n}^{(i)}\}_{i=1}^{300}$ 。每次计算梯度都是按照 VQE 的解析梯度公式计算关于第一个参数 $\theta_{1,1}$ 的偏导数。然后统计得到的这 300 个梯度的平均值和方差。其中解析梯度的公式为：

$$\partial\theta_j \equiv \frac{\partial\mathcal{L}}{\partial\theta_j} = \frac{1}{2} \left[\mathcal{L}(\theta_j + \frac{\pi}{2}) - \mathcal{L}(\theta_j - \frac{\pi}{2}) \right]$$

具体推导请参见：[arXiv:1803.00745](https://arxiv.org/abs/1803.00745)

```

1  # 超参数设置
2  np.random.seed(42)          # 固定 Numpy 的随机种子
3  N = 2                        # 设置量子比特数量
4  samples = 300               # 设定采样随机网络结构的数量
5  THETA_SIZE = N              # 设置参数 theta 的大小
6  ITR = 1                     # 设置迭代次数
7  LR = 0.2                    # 设定学习速率
8  SEED = 1                    # 固定优化器中随机初始化的种子
9
10 # 初始化梯度数值的寄存器
11 grad_info = []
12
13 class manual_gradient(fluid.dygraph.Layer):
14
15     # 初始化一个可学习参数列表, 并用 [0, 2*pi] 的均匀分布来填充初始值
16     def __init__(self, shape, param_attr=fluid.initializer.Uniform(
17         low=0.0, high=2 * np.pi, seed=1), dtype='float64'):
18         super(manual_gradient, self).__init__()
19
20         # 将 Numpy array 转换成 Paddle 动态图模式中支持的 variable
21         self.H = fluid.dygraph.to_variable(density_op(N))
22
23     # 定义损失函数和前向传播机制
24     def forward(self):
25
26         # 初始化三个 theta 参数列表
27         theta_np = np.random.uniform(
28             low=0., high= 2 * np.pi, size=(THETA_SIZE))
29         theta_plus_np = np.copy(theta_np)
30         theta_minus_np = np.copy(theta_np)
31
32         # 修改用以计算解析梯度
33         theta_plus_np[0] += np.pi/2
34         theta_minus_np[0] -= np.pi/2
35
36         # 将 Numpy array 转换成 Paddle 动态图模式中支持的 variable
37         theta = fluid.dygraph.to_variable(theta_np)
38         theta_plus = fluid.dygraph.to_variable(theta_plus_np)
39         theta_minus = fluid.dygraph.to_variable(theta_minus_np)
40
41         # 生成随机目标, 在 rand_circuit 中随机选取电路门
42         target = np.random.choice(3, N)
43
44         U = rand_circuit(theta, target, N)
45         U_dagger = hermitian(U)
46         U_plus = rand_circuit(theta_plus, target, N)
47         U_plus_dagger = hermitian(U_plus)
48         U_minus = rand_circuit(theta_minus, target, N)
49         U_minus_dagger = hermitian(U_minus)
50
51         # 计算解析梯度
52         grad = (matmul(matmul(U_plus_dagger,

```

```

53         self.H), U_plus).real[0][0]
54         - matmul(matmul(U_minus_dagger,
55                         self.H), U_minus).real[0][0])/2
56     return grad
57
58 # 定义主程序段
59 def main():
60
61     # 初始化paddle动态图机制
62     with fluid.dygraph.guard():
63
64         # 设置QNN的维度
65         sampling = manual_gradient(shape=[THETA_SIZE])
66
67         # 采样获得梯度信息
68         grad = sampling()
69
70     return grad.numpy()
71
72
73 # 记录运行时间
74 time_start = time.time()
75
76 # 开始采样
77 for i in range(samples):
78     if __name__ == '__main__':
79         grad = main()
80         grad_info.append(grad)
81
82 time_span = time.time() - time_start
83 print('主程序段总共运行了', time_span, '秒')
84 print("采样", samples, "个随机网络关于第一个参数梯度的均值是：",
      np.mean(grad_info))
85 print("采样", samples, "个随机网络关于第一个参数梯度的方差是：",
      np.var(grad_info))

```

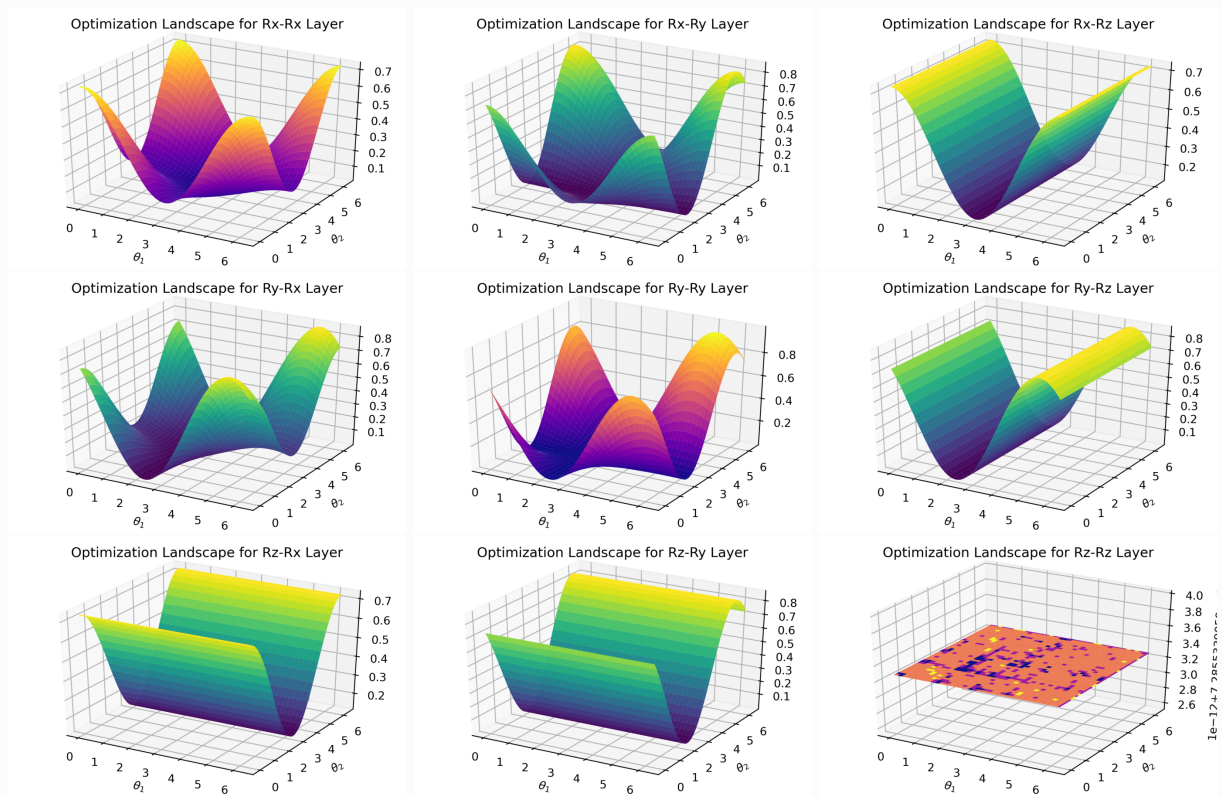
```

1 主程序段总共运行了 9.175814151763916 秒
2 采样 300 个随机网络关于第一个参数梯度的均值是： 0.005925709445960606
3 采样 300 个随机网络关于第一个参数梯度的方差是： 0.028249053148446363

```

优化曲面的可视化

接下来我们试着利用 Matplotlib来可视化这个优化曲面（Optimization landscape）。在两个量子比特 的情况下，我们只有两个参数 θ_1 和 θ_2 ，并且第二层的随机电路电路总共有9种情况。这个很容易画出来：



可以看到的是最后一张图中 R_z - R_z 结构所展示出的平原结构是我们非常不想见到的。在这种情况下，我们不能收敛到理论最小值。如果你想自己试着画一些优化曲面，不妨参见以下代码：

```
1  # 引入必要的 package
2  from matplotlib import cm
3  from mpl_toolkits.mplot3d import Axes3D
4  from matplotlib.ticker import LinearLocator, FormatStrFormatter
5
6  time_start = time.time()
7  N = 2
8
9  # 设置图像比例 竖：横 = 0.3
10 fig = plt.figure(figsize=plt.figaspect(0.3))
11
12 # 生成 x, y 轴上的点
13 X = np.linspace(0, 2*np.pi, 80)
14 Y = np.linspace(0, 2*np.pi, 80)
15
16 # 生成 2D 网格 (mesh)
17 xx, yy = np.meshgrid(X, Y)
```

```

18
19
20 # 定义必要的逻辑门
21 def rx(theta):
22     mat = np.array([[np.cos(theta/2), -1j*np.sin(theta/2)],
23                     [-1j*np.sin(theta/2), np.cos(theta/2)]])
24     return mat
25
26 def ry(theta):
27     mat = np.array([[np.cos(theta/2), -1*np.sin(theta/2)],
28                     [np.sin(theta/2), np.cos(theta/2)]])
29     return mat
30
31 def rz(theta):
32     mat = np.array([[np.exp(-1j*theta/2), 0],
33                     [0, np.exp(1j*theta/2)]])
34     return mat
35
36 def CNOT():
37     mat = np.array([[1,0,0,0],[0,1,0,0],[0,0,0,1],[0,0,1,0]])
38     return mat
39
40 # ===== 第一张图 =====
41 # 我们可视化第二层是 kron(Ry, Ry) 的情况
42 ax = fig.add_subplot(1, 2, 1, projection='3d')
43
44 # 向前传播计算损失函数:
45 def cost_yy(para):
46     L1 = np.kron(ry(np.pi/4), ry(np.pi/4))
47     L2 = np.kron(ry(para[0]), ry(para[1]))
48     U = np.matmul(np.matmul(L1, L2), CNOT())
49     H = np.zeros((2 ** N, 2 ** N))
50     H[0, 0] = 1
51     val = (U.conj().T @ H @ U).real[0][0]
52     return val
53
54 # 画出图像
55 Z = np.array([[cost_yy([x, y]) for x in X] for y in
56 Y]).reshape(len(Y), len(X))
57 surf = ax.plot_surface(xx, yy, Z, cmap='plasma')
58 #cset = ax.contourf(xx, yy, Z, zdir='z', offset=np.min(Z),
59 cmap='viridis')
60 ax.set_xlabel(r"$\theta_1$")
61 ax.set_ylabel(r"$\theta_2$")
62 ax.set_title("Optimization Landscape for Ry-Ry Layer")
63 #fig.colorbar(surf, shrink=0.5, aspect=5)
64
65 # ===== 第二张图 =====
66 # 我们可视化第二层是 kron(Rx, Rz) 的情况
67 ax = fig.add_subplot(1, 2, 2, projection='3d')

```

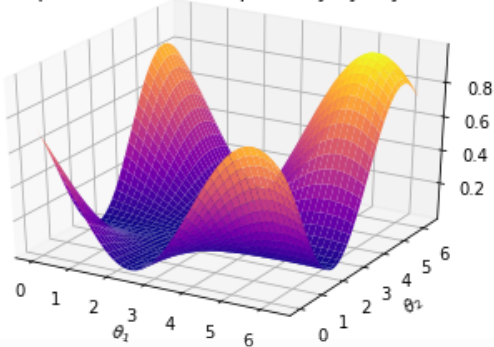


```

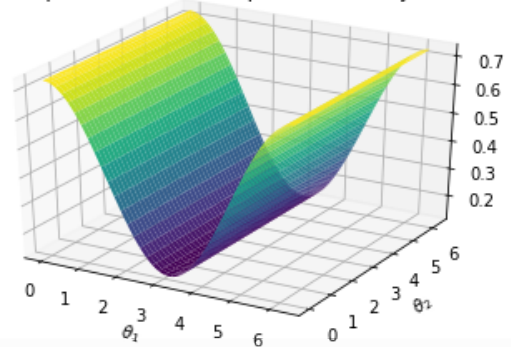
68 def cost_xz(para):
69     L1 = np.kron(ry(np.pi/4), ry(np.pi/4))
70     L2 = np.kron(rx(para[0]), rz(para[1]))
71     U = np.matmul(np.matmul(L1, L2), CNOT())
72     H = np.zeros((2 ** N, 2 ** N))
73     H[0, 0] = 1
74     val = (U.conj().T @ H @ U).real[0][0]
75     return val
76
77 Z = np.array([[cost_xz([x, y]) for x in X] for y in
78 Y]).reshape(len(Y), len(X))
79 surf = ax.plot_surface(xx, yy, Z, cmap='viridis')
80 #cset = ax.contourf(xx, yy, Z, zdir='z', offset=np.min(Z),
81 cmap='viridis')
82 ax.set_xlabel(r"$\theta_1$")
83 ax.set_ylabel(r"$\theta_2$")
84 ax.set_title("Optimization Landscape for Rx-Rz Layer")
85
86 plt.show()
87
88 time_span = time.time() - time_start
89 print('主程序段总共运行了', time_span, '秒')

```

Optimization Landscape for Ry-Ry Layer



Optimization Landscape for Rx-Rz Layer



1 主程序段总共运行了 2.8104310035705566 秒

更多的量子比特

接着我们再看看不断的增加量子比特的数量，会对我们的梯度带来什么影响。

```
1  # 超参数设置
2  selected_qubit = [2, 4, 6, 8]
3  samples = 300
4  grad_val = []
5  means, variances = [], []
6
7
8  # 记录运算时长
9  time_start = time.time()
10
11 # 不断增加量子比特数量
12 for N in selected_qubit:
13     grad_info = []
14     THETA_SIZE = N
15     for i in range(samples):
16         class manual_gradient(fluid.dygraph.Layer):
17             # 初始化一个长度为 THETA_SIZE 的可学习参数列表
18             def __init__(self, shape, param_attr=
19                 fluid.initializer.Uniform(low=0.0, high=2 * np.pi,
20                 seed=1), dtype='float64'):
21                 super(manual_gradient, self).__init__()
22
23             # 转换成 Paddle 动态图模式中支持的 variable
24             self.H = fluid.dygraph.to_variable(
25                 density_op(N))
26
27             # 定义损失函数和前向传播机制
28             def forward(self):
29
30                 # 初始化三个 theta 参数列表
31                 theta_np = np.random.uniform(
32                     low=0., high= 2 * np.pi, size=(THETA_SIZE))
33                 theta_plus_np = np.copy(theta_np)
34                 theta_minus_np = np.copy(theta_np)
35
36                 # 修改用以计算解析梯度
37                 theta_plus_np[0] += np.pi/2
38                 theta_minus_np[0] -= np.pi/2
39
40                 # 转换成 Paddle 动态图模式中支持的 variable
41                 theta = fluid.dygraph.to_variable(theta_np)
42                 theta_plus = fluid.dygraph.to_variable(
43                     theta_plus_np)
44                 theta_minus = fluid.dygraph.to_variable(
45                     theta_minus_np)
46
```

```

47         # 生成随机目标, 在 rand_circuit 中随机选取电路门
48         target = np.random.choice(3, N)
49
50         U = rand_circuit(theta, target, N)
51         U_dagger = hermitian(U)
52         U_plus = rand_circuit(theta_plus,
53                                 target, N)
54         U_plus_dagger = hermitian(U_plus)
55         U_minus = rand_circuit(theta_minus,
56                                 target, N)
57         U_minus_dagger = hermitian(U_minus)
58
59
60         # 计算解析梯度
61         grad = ( matmul(matmul(U_plus_dagger,
62                                self.H), U_plus).real[0][0]
63                 - matmul(matmul(U_minus_dagger,
64                                self.H), U_minus).real[0][0])/2
65
66         return grad
67
68
69     # 定义主程序段
70     def main():
71
72         # 初始化paddle动态图机制
73         with fluid.dygraph.guard():
74
75             sampling = manual_gradient(shape=[THETA_SIZE])
76
77             # 采样获得梯度信息
78             grad = sampling()
79
80             return grad.numpy()
81
82         if __name__ == '__main__':
83             grad = main()
84             grad_info.append(grad)
85
86         # 记录采样信息
87         grad_val.append(grad_info)
88         means.append(np.mean(grad_info))
89         variances.append(np.var(grad_info))

```

```

1 grad = np.array(grad_val)
2 means = np.array(means)
3 variances = np.array(variances)
4 n = np.array(selected_qubit)
5 print("我们接着画出这个采样出来的梯度的统计结果：")

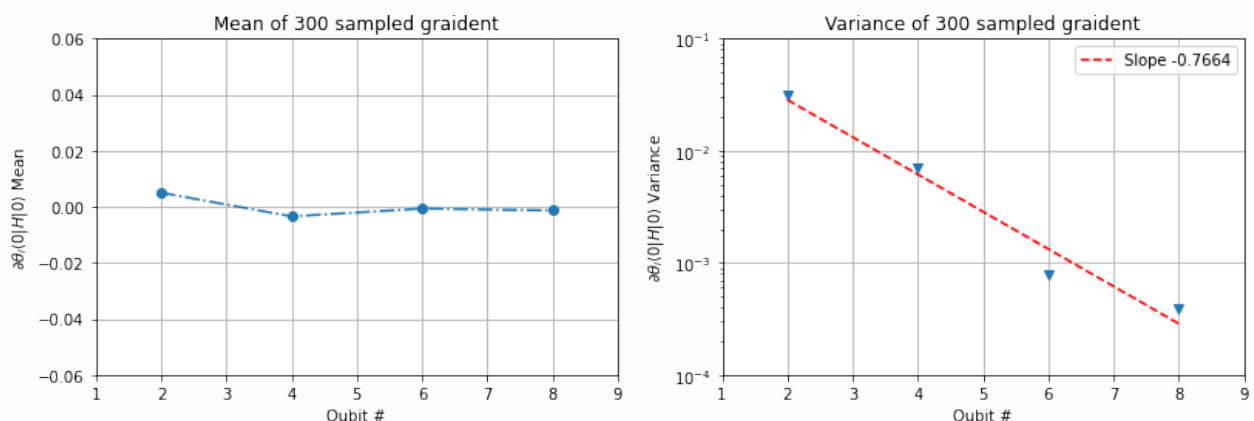
```

```

6 fig = plt.figure(figsize=plt.figaspect(0.3))
7
8
9 # ===== 第一张图 =====
10 # 统计出随机采样的梯度平均值和量子比特数量的关系
11 plt.subplot(1, 2, 1)
12 plt.plot(n, means, "o-.")
13 plt.xlabel(r"Qubit #")
14 plt.ylabel(r"$\partial \langle H \rangle / \partial \theta_i$ Mean")
15 plt.title("Mean of {} sampled gradient".format(samples))
16 plt.xlim([1,9])
17 plt.ylim([-0.06, 0.06])
18 plt.grid()
19
20 # ===== 第二张图 =====
21 # 统计出随机采样的梯度的方差和量子比特数量的关系
22 plt.subplot(1, 2, 2)
23 plt.semilogy(n, variances, "v")
24
25 # 多项式拟合
26 fit = np.polyfit(n, np.log(variances), 1)
27 slope = fit[0]
28 intercept = fit[1]
29 plt.semilogy(n, np.exp(n*slope + intercept), "r--", label="Slope
30 {:03.4f}".format(slope))
31 plt.xlabel(r"Qubit #")
32 plt.ylabel(r"$\partial \langle H \rangle / \partial \theta_i$ Variance")
33 plt.title("Variance of {} sampled gradient".format(samples))
34 plt.legend()
35 plt.xlim([1,9])
36 plt.ylim([0.0001, 0.1])
37 plt.grid()
38 plt.show()

```

1 我们接着画出这个采样出来的梯度的统计结果：



要注意的是，在理论上，只有当我们选取的网络结构还有损失函数满足一定条件时 (2-design) 详见论文 (1), 才会出现这种效应。接着我们不妨可视化一下不同量子比特数量对的优化曲面的影响：

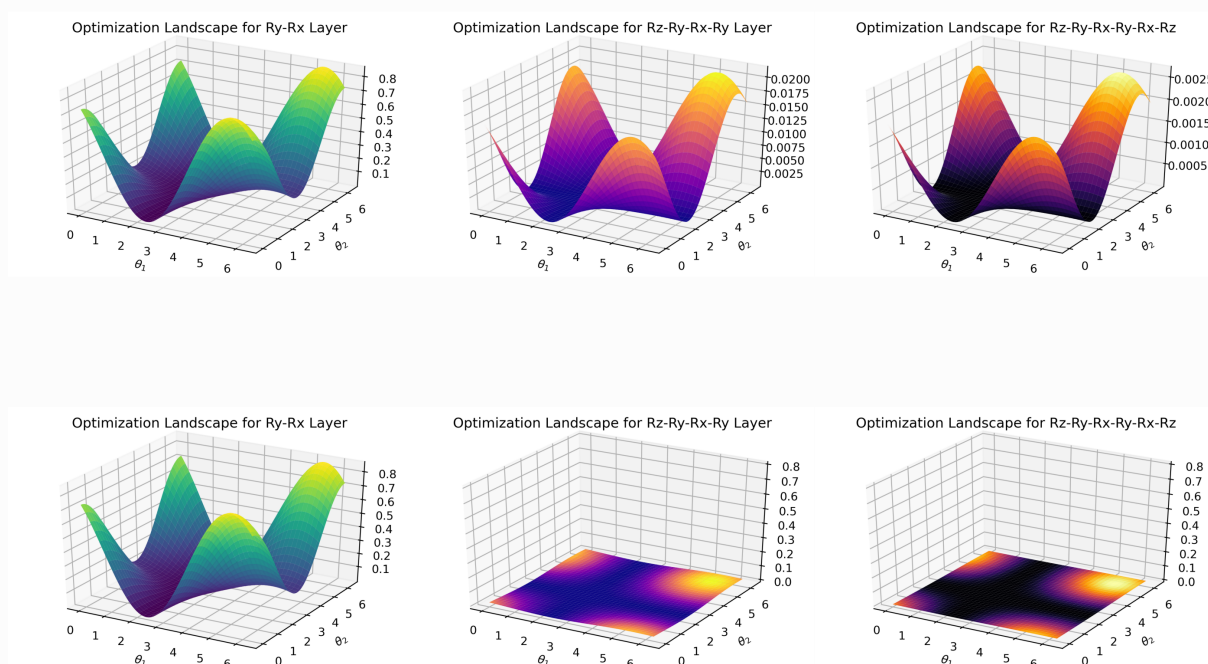


图 1. (a). 2-量子比特 (b). 4-量子比特 (c). 6-量子比特

画图时 θ_1 和 θ_2 是前两个电路参数, 剩余参数全部固定为 π 。不然我们画不出这个高维度的流形。结果不出所料, 陡峭程度随着 n 的增大越来越小了, 注意到 **Z** 轴尺度的极速减小。相对于2量子比特的情况, 6量子比特的优化曲面已经非常扁平了。

参考文献

- (1) McClean, J. R., Boixo, S., Smelyanskiy, V. N., Babbush, R. & Neven, H. Barren plateaus in quantum neural network training landscapes. Nat. Commun. 9, 4812 (2018).
- (2) Cerezo, M., Sone, A., Volkoff, T., Cincio, L. & Coles, P. J. Cost-Function-Dependent Barren Plateaus in Shallow Quantum Neural Networks. arXiv:2001.00550 (2020).