File - performance_test

```
C:\Users\usuario\Anaconda3\envs\tfq\python.exe C:/Users/usuario/qGAN/quantumGAN/performance_test.py
[0.6945839  0.52642846 0.29759293 0.13430657 0.74127211 0.19292928]
Epoch 0: Loss: [-0.59065402] [0.4183018  0.         0.40034457 0.        ] [0.34912109 0.1171875  0.49902344 0.03466797]
[0.92787364] [0.92900903]
Timer unit: 1e-07 s

Total time: 34.2066 s
File: C:/Users/usuario/qGAN/quantumGAN/performance_test.py
Function: main at line 12

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    12                                           def main():
    13
    14         1         35.0     35.0      0.0     seed = 71
    15         1         45.0     45.0      0.0     np.random.seed = seed
    16
    17         1        132.0    132.0      0.0     bounds = np.array([0., 3.])
    18         1         24.0     24.0      0.0     num_qubits = [2]
    19         1         24.0     24.0      0.0     k = len(num_qubits)
    20         1         21.0     21.0      0.0     batch_size = 10
    21         1         24.0     24.0      0.0     entangler_map = [[0, 1]]
    22
    23         1        526.0    526.0      0.0     randoms = np.random.normal(-np.pi * .01, np.pi * .01, 2)
    24
    25         1       2120.0   2120.0      0.0     init_dist = qiskit.QuantumCircuit(2)
    26         1       1462.0   1462.0      0.0     init_dist.ry(randoms[0], 0)
    27         1        790.0    790.0      0.0     init_dist.ry(randoms[1], 1)
    28
    29         1   26841604.0 26841604.0      7.8   ansatz = TwoLocal(int(np.sum(num_qubits)), 'rx', 'cz', entanglement=
entangler_map, reps=2, insert_barriers=True)
    30
    31         1       2210.0   2210.0      0.0     init_params = np.random.rand(ansatz.num_parameters_settable)
    32
    33         1         28.0     28.0      0.0     train_data = []
    34        16        368.0     23.0      0.0     for _ in range(15):
    35        15       1618.0    107.9      0.0         x2 = np.random.uniform(.5, .4, (2,))
    36        15       1296.0     86.4      0.0         fake_datapoint = np.random.uniform(-np.pi * .01, np.pi * .01, (2
,))
    37        15       1057.0     70.5      0.0         real_datapoint = np.array([x2[1], 0., x2[0], 0])
    38        15        529.0     35.3      0.0         train_data.append((real_datapoint, fake_datapoint))
    39
    40         1     155648.0 155648.0      0.0     g_circuit = ansatz.compose(init_dist, front=True)
    41
    42         1         40.0     40.0      0.0     discriminator = Network(training_data=train_data,
    43         1         23.0     23.0      0.0                             mini_batch_size=batch_size,
    44         1         26.0     26.0      0.0                             sizes=[4, 16, 8, 1],
    45         1        942.0    942.0      0.0                             loss_BCE=True)
    46         1         27.0     27.0      0.0     generator = PerformanceQuantumGenerator(training_data=train_data,
    47         1         22.0     22.0      0.0                                 mini_batch_size=batch_size,
    48         1         24.0     24.0      0.0                                 num_qubits=num_qubits,
    49         1         21.0     21.0      0.0                                 generator_circuit=g_circuit,
    50         1         22.0     22.0      0.0                                 shots=2048,
    51         1       6398.0   6398.0      0.0                                 learning_rate=.1)
    52         1         52.0     52.0      0.0     generator.set_discriminator(discriminator)
    53
    54         1         22.0     22.0      0.0     num_epochs = 1
    55
    56         1         29.0     29.0      0.0     def train():
    57                                                 for o in range(num_epochs):
    58                                                     mini_batches = discriminator.create_mini_batches()
    59                                                     for mini_batch in mini_batches:
    60                                                         output_real = mini_batch[0][0]
    61                                                         output_fake = generator.get_output(latent_space_noise=
mini_batch[0][1],
    62                                                                                            params=None)
    63                                                         generator.set_mini_batch(mini_batch)
    64                                                         generator.shots = 2048
    65                                                         lp_wrapper_gen = lp(generator.train_mini_batch)
    66                                                         lp_wrapper_gen()
    67                                                         discriminator.train_mini_batch(generator.mini_batch, .1
, o)
    68                                                     print("Epoch {}: Loss: {}".format(o, discriminator.ret["loss
"][-1]), output_real, output_fake)
    69                                                     print(discriminator.ret["label real"][-1], discriminator.ret
["label fake"][-1])
    70
    71         1        479.0    479.0      0.0     lp_wrapper_train = lp(train)
    72         1  315048200.0 315048200.0     92.1     lp_wrapper_train()

Total time: 31.5048 s
File: C:/Users/usuario/qGAN/quantumGAN/performance_test.py
Function: train at line 56

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    56                                           def train():
    57         2         38.0     19.0      0.0     for o in range(num_epochs):
    58         1        804.0    804.0      0.0         mini_batches = discriminator.create_mini_batches()
```

File - performance_test

```
 84     59         3         89.0     29.7      0.0             for mini_batch in mini_batches:
 85     60         2         26.0     13.0      0.0                 output_real = mini_batch[0][0]
 86     61         2         36.0     18.0      0.0                 output_fake = generator.get_output(latent_space_noise=
    mini_batch[0][1],
 87     62         2   12104474.0 6052237.0      3.8                                                     params=None)
 88     63         2        187.0     93.5      0.0                 generator.set_mini_batch(mini_batch)
 89     64         2         21.0     10.5      0.0                 generator.shots = 2048
 90     65         2       1165.0    582.5      0.0                 lp_wrapper_gen = lp(generator.train_mini_batch)
 91     66         2  302831538.0 151415769.0     96.1                     lp_wrapper_gen()
 92     67         2      89653.0  44826.5      0.0                 discriminator.train_mini_batch(generator.mini_batch, .1
    , o)
 93     68         1      13233.0  13233.0      0.0             print("Epoch {}: Loss: {}".format(o, discriminator.ret["
    loss"][-1]), output_real, output_fake)
 94     69         1       6328.0   6328.0      0.0             print(discriminator.ret["label real"][-1], discriminator.
    ret["label fake"][-1])
 95
 96 Total time: 30.2808 s
 97 File: C:\Users\usuario\qGAN\quantumGAN\performance_quantum_generator.py
 98 Function: train_mini_batch at line 118
 99
100 Line #      Hits         Time  Per Hit   % Time  Line Contents
101 ==============================================================
102    118                                           def train_mini_batch(self):
103    119         2        153.0     76.5      0.0         nabla_theta = np.zeros(self.parameter_values.shape)
104    120         2         22.0     11.0      0.0         new_images = []
105    121
106    122        17        950.0     55.9      0.0         for _, noise in self.mini_batch:
107    123       105       3096.0     29.5      0.0             for index in range(len(self.parameter_values)):
108    124        90      12465.0    138.5      0.0                 perturbation_vector = np.zeros(len(self.
    parameter_values))
109    125        90       2573.0     28.6      0.0                 perturbation_vector[index] = 1
110    126
111    127        90      10282.0    114.2      0.0                 pos_params = self.parameter_values + (np.pi / 4) *
    perturbation_vector
112    128        90       5960.0     66.2      0.0                 neg_params = self.parameter_values - (np.pi / 4) *
    perturbation_vector
113    129
114    130        90  140881590.0 1565351.0     46.5                 pos_result = self.get_output(noise, params=pos_params)
115    131        90  138590021.0 1539889.1     45.8                 neg_result = self.get_output(noise, params=neg_params)
116    132
117    133        90     113842.0   1264.9      0.0                 pos_result = self.discriminator.predict(pos_result)
118    134        90      56180.0    624.2      0.0                 neg_result = self.discriminator.predict(neg_result)
119    135        90      59406.0    660.1      0.0                 gradient = self.BCE(pos_result, np.array([1.])) - self.
    BCE(neg_result, np.array([1.]))
120    136        90      28771.0    319.7      0.0                 nabla_theta[index] += gradient
121    137        15   23041791.0 1536119.4      7.6             new_images.append(self.get_output(noise))
122    138
123    139        14        286.0     20.4      0.0         for index in range(len(self.parameter_values)):
124    140        12        488.0     40.7      0.0             self.parameter_values[index] -= (self.learning_rate / self.
    mini_batch_size) * nabla_theta[index]
125    141
126    142         2        323.0    161.5      0.0         self.mini_batch = [(datapoint[0], fake_image) for datapoint,
    fake_image in zip(self.mini_batch, new_images)]
127
128
129 Process finished with exit code 0
130
```