QGAN

June 19, 2019

1 Quantum Generative Adversarial Network

1.1 Authors

Michel Barbeau
Joaquin Garcia-Alfaro
Version: June 19, 2019
Reuse code from:
Example Q3 - Variational classifier - 2. Iris classification
Example Q4 - Quantum Generative Adversarial Network
Basic tutorial: Gaussian transformation

1.2 1. Preamble

PennyLane and other relevant package are imported.

```
In [1]: import pennylane as qml
        from pennylane import numpy as np
        from pennylane.optimize import GradientDescentOptimizer
        import math
```

A six-qubit device is create. With probability amplitude encoding, up to 2^6 single scalar values can be reprented in probability amplitudes in the input circuit quantum state.

```
In [2]: num_qubits = 6
        dev = qml.device('default.qubit', wires=num_qubits)
```

1.3 2. Pre-processing of real data

Read the raw real data, assuming m input files. Each file may contain up to 2^6 single scalar values. Each line comprises a timestamp (not used), a x-velocity, a y-velocity and a z-velocity,

```
In [3]: m = 6 # number of real data files
        real_values = np.zeros((m,2**num_qubits))
        for i in range(m):
            # read real data from one file
            data = np.loadtxt('Parrot_Mambo_Data/realdata'+str(i)+'.txt')
            selected_data = data[:,1:4] # omit the timestamp
```

```
            # reshape three-column matrix into an array
            real_values[i][0:len(selected_data)*3] = \
                selected_data.reshape(len(selected_data)*3)
            print("Raw data: ",real_values[i,:])
```

```
Raw data:  [-0.08376908  0.33940566  0.00183541 -0.08376908  0.33940566  0.00183541
 -0.08376908  0.33940566  0.00183541  0.00109504 -0.01502317  0.01236341
  0.00109504 -0.01502317  0.01236341  0.00109504 -0.01502317  0.01236341
 -0.03160943  0.01453508  0.01444724 -0.03160943  0.01453508  0.01444724
 -0.03160943  0.01453508  0.01444724 -0.03160943  0.01453508  0.01444724
 -0.02090058  0.01354719  0.00997251 -0.02090058  0.01354719  0.00997251
 -0.02090058  0.01354719  0.00997251 -0.00556759 -0.00805624  0.01183656
 -0.00556759 -0.00805624  0.01183656 -0.00556759 -0.00805624  0.01183656
  0.01629235 -0.00671357  0.01200437  0.01629235 -0.00671357  0.01200437
  0.01629235 -0.00671357  0.01200437  0.01629235 -0.00671357  0.01200437
  0.01775298  0.00872391  0.01240711  0.        ]
Raw data:  [ 0.07393985 -0.08155741 -0.00174763 -0.01699297  0.0439186  -0.00723943
 -0.01699297  0.0439186  -0.00723943 -0.0467293   0.02906708  0.02376857
 -0.0467293   0.02906708  0.02376857  0.00972683  0.05553944  0.01813706
  0.00972683  0.05553944  0.01813706  0.00972683  0.05553944  0.01813706
  0.00267601 -0.00370482  0.01482126  0.00267601 -0.00370482  0.01482126
 -0.01471184 -0.0770198  -0.00371389 -0.01471184 -0.0770198  -0.00371389
 -0.01471184 -0.0770198  -0.00371389  0.01513966  0.00751151  0.0016375
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.        ]
Raw data:  [ 9.180749e-02 -7.005025e-02  1.178688e-02  9.180749e-02 -7.005025e-02
  1.178688e-02  1.077333e-02 -1.144817e-02  1.415802e-02  1.077333e-02
 -1.144817e-02  1.415802e-02  1.077333e-02 -1.144817e-02  1.415802e-02
 -5.388897e-02  2.071080e-02  1.527728e-02 -5.388897e-02  2.071080e-02
  1.527728e-02 -3.625565e-02  2.723628e-02  1.610531e-02 -3.625565e-02
  2.723628e-02  1.610531e-02 -3.625565e-02  2.723628e-02  1.610531e-02
  9.066000e-05 -1.490826e-02 -2.906900e-04  9.066000e-05 -1.490826e-02
 -2.906900e-04  9.066000e-05 -1.490826e-02 -2.906900e-04  3.568355e-02
 -3.052060e-02  2.274893e-02  3.568355e-02 -3.052060e-02  2.274893e-02
  3.568355e-02 -3.052060e-02  2.274893e-02 -7.671050e-03  3.214242e-02
  9.254490e-03  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00]
Raw data:  [-0.00509642 -0.00832376  0.01506707 -0.00509642 -0.00832376  0.01506707
 -0.00509642 -0.00832376  0.01506707 -0.02719803  0.03381     0.01706991
 -0.02719803  0.03381     0.01706991  0.00247172  0.00940448  0.01620432
  0.00247172  0.00940448  0.01620432  0.00247172  0.00940448  0.01620432
 -0.01173865 -0.00884636  0.01537707 -0.01173865 -0.00884636  0.01537707
 -0.01173865 -0.00884636  0.01537707  0.01738478 -0.01649329  0.01408078
  0.01738478 -0.01649329  0.01408078  0.01618808 -0.0008779   0.016907
  0.          0.          0.          0.          0.          0.
```

```
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.        ]
Raw data:  [-9.5720300e-03  2.7768114e-01  8.9101400e-03 -9.5720300e-03
  2.7768114e-01  8.9101400e-03 -9.5720300e-03  2.7768114e-01
  8.9101400e-03  2.2996000e-02 -1.3513980e-02  8.8767600e-03
  2.2996000e-02 -1.3513980e-02  8.8767600e-03 -1.3623320e-02
  6.5740600e-03  1.1207880e-02 -1.3623320e-02  6.5740600e-03
  1.1207880e-02 -1.3623320e-02  6.5740600e-03  1.1207880e-02
 -2.0980570e-02  2.6495010e-02  1.3232210e-02 -2.0980570e-02
  2.6495010e-02  1.3232210e-02 -2.0980570e-02  2.6495010e-02
  1.3232210e-02  5.8310000e-05  1.8326290e-02  1.0773430e-02
  5.8310000e-05  1.8326290e-02  1.0773430e-02  1.3626400e-03
  2.7880800e-03  1.2730800e-02  1.3626400e-03  2.7880800e-03
  1.2730800e-02  1.3626400e-03  2.7880800e-03  1.2730800e-02
  5.7880700e-03 -1.5020210e-02  1.2070470e-02  5.7880700e-03
 -1.5020210e-02  1.2070470e-02 -5.3494500e-03 -3.8512300e-03
  1.1064610e-02  0.0000000e+00  0.0000000e+00  0.0000000e+00
  0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00]
Raw data:  [-0.00068315 -0.03387602  0.01362513 -0.00068315 -0.03387602  0.01362513
 -0.00068315 -0.03387602  0.01362513 -0.03181319  0.02339145  0.01939554
 -0.03181319  0.02339145  0.01939554 -0.0176115   0.01941477  0.01413892
 -0.0176115   0.01941477  0.01413892 -0.0176115   0.01941477  0.01413892
 -0.00012692  0.01502633  0.00993828 -0.00012692  0.01502633  0.00993828
 -0.00012692  0.01502633  0.00993828  0.00801174 -0.00790758  0.01288757
  0.00801174 -0.00790758  0.01288757  0.00737117  0.00776267  0.01545549
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.        ]
```

This example uses probability amplitude encoding, which requires normalized data. Let $x_0, \ldots, x_{n-1}$ be the real data values, their normal form is

$$x_0/m, \ldots, x_{n-1}/m$$

where

$$m = \sqrt{x_0^2 + \ldots + x_{n-1}^2}.$$

```
In [4]: # Normalization function
        def normalize(values):
            m = np.sqrt(np.sum(values ** 2))
            return values / m
        for i in range(len(real_values)):
           real_values[i,:] = normalize(real_values[i,:])
           # check consistency of probability amplitudes
           assert np.abs(1-np.sum(real_values[i,:] ** 2))<=0.0000001
        print("Normalized data: ",real_values)
```

```
Normalized data:  [[-1.36124893e-01  5.51534757e-01  2.98254428e-03 -1.36124893e-01
   5.51534757e-01  2.98254428e-03 -1.36124893e-01  5.51534757e-01
   2.98254428e-03  1.77944181e-03 -2.44126760e-02  2.00905616e-02
   1.77944181e-03 -2.44126760e-02  2.00905616e-02  1.77944181e-03
  -2.44126760e-02  2.00905616e-02 -5.13653759e-02  2.36195290e-02
   2.34767888e-02 -5.13653759e-02  2.36195290e-02  2.34767888e-02
  -5.13653759e-02  2.36195290e-02  2.34767888e-02 -5.13653759e-02
   2.36195290e-02  2.34767888e-02 -3.39634770e-02  2.20142061e-02
   1.62053452e-02 -3.39634770e-02  2.20142061e-02  1.62053452e-02
  -3.39634770e-02  2.20142061e-02  1.62053452e-02 -9.04734293e-03
  -1.30914033e-02  1.92344295e-02 -9.04734293e-03 -1.30914033e-02
   1.92344295e-02 -9.04734293e-03 -1.30914033e-02  1.92344295e-02
   2.64750956e-02 -1.09095623e-02  1.95071211e-02  2.64750956e-02
  -1.09095623e-02  1.95071211e-02  2.64750956e-02 -1.09095623e-02
   1.95071211e-02  2.64750956e-02 -1.09095623e-02  1.95071211e-02
   2.88486218e-02  1.41763681e-02  2.01615742e-02  0.00000000e+00]
 [ 3.19283423e-01 -3.52177196e-01 -7.54652991e-03 -7.33782072e-02
   1.89647138e-01 -3.12609505e-02 -7.33782072e-02  1.89647138e-01
  -3.12609505e-02 -2.01784165e-01  1.25516035e-01  1.02636270e-01
  -2.01784165e-01  1.25516035e-01  1.02636270e-01  4.20019189e-02
   2.39827678e-01  7.83185604e-02  4.20019189e-02  2.39827678e-01
   7.83185604e-02  4.20019189e-02  2.39827678e-01  7.83185604e-02
   1.15554148e-02 -1.59979715e-02  6.40004359e-02  1.15554148e-02
  -1.59979715e-02  6.40004359e-02 -6.35279439e-02 -3.32583112e-01
  -1.60371371e-02 -6.35279439e-02 -3.32583112e-01 -1.60371371e-02
  -6.35279439e-02 -3.32583112e-01 -1.60371371e-02  6.53753352e-02
   3.24358330e-02  7.07097196e-03  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 4.01126866e-01 -3.06064758e-01  5.14994390e-02  4.01126866e-01
  -3.06064758e-01  5.14994390e-02  4.70710189e-02 -5.00195414e-02
   6.18594647e-02  4.70710189e-02 -5.00195414e-02  6.18594647e-02
   4.70710189e-02 -5.00195414e-02  6.18594647e-02 -2.35452615e-01
   9.04899839e-02  6.67497548e-02 -2.35452615e-01  9.04899839e-02
   6.67497548e-02 -1.58408810e-01  1.19001223e-01  7.03675977e-02
  -1.58408810e-01  1.19001223e-01  7.03675977e-02 -1.58408810e-01
   1.19001223e-01  7.03675977e-02  3.96113233e-04 -6.51374262e-02
  -1.27008775e-03  3.96113233e-04 -6.51374262e-02 -1.27008775e-03
   3.96113233e-04 -6.51374262e-02 -1.27008775e-03  1.55909181e-01
  -1.33351131e-01  9.93950166e-02  1.55909181e-01 -1.33351131e-01
   9.93950166e-02  1.55909181e-01 -1.33351131e-01  9.93950166e-02
  -3.35164837e-02  1.40437215e-01  4.04348770e-02  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

```
[-5.13822540e-02 -8.39203893e-02  1.51906636e-01 -5.13822540e-02
 -8.39203893e-02  1.51906636e-01 -5.13822540e-02 -8.39203893e-02
  1.51906636e-01 -2.74211326e-01  3.40873399e-01  1.72099327e-01
 -2.74211326e-01  3.40873399e-01  1.72099327e-01  2.49199526e-02
  9.48162397e-02  1.63372423e-01  2.49199526e-02  9.48162397e-02
  1.63372423e-01  2.49199526e-02  9.48162397e-02  1.63372423e-01
 -1.18349409e-01 -8.91892577e-02  1.55032065e-01 -1.18349409e-01
 -8.91892577e-02  1.55032065e-01 -1.18349409e-01 -8.91892577e-02
  1.55032065e-01  1.75273855e-01 -1.66285827e-01  1.41962832e-01
  1.75273855e-01 -1.66285827e-01  1.41962832e-01  1.63208691e-01
 -8.85101322e-03  1.70456864e-01  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[-1.95053898e-02  5.65844327e-01  1.81566244e-02 -1.95053898e-02
  5.65844327e-01  1.81566244e-02 -1.95053898e-02  5.65844327e-01
  1.81566244e-02  4.68600646e-02 -2.75380925e-02  1.80886044e-02
  4.68600646e-02 -2.75380925e-02  1.80886044e-02 -2.77608999e-02
  1.33962809e-02  2.28388407e-02 -2.77608999e-02  1.33962809e-02
  2.28388407e-02 -2.77608999e-02  1.33962809e-02  2.28388407e-02
 -4.27531251e-02  5.39901670e-02  2.69639161e-02 -4.27531251e-02
  5.39901670e-02  2.69639161e-02 -4.27531251e-02  5.39901670e-02
  2.69639161e-02  1.18821115e-04  3.73443700e-02  2.19535409e-02
  1.18821115e-04  3.73443700e-02  2.19535409e-02  2.77671762e-03
  5.68140585e-03  2.59421686e-02  2.77671762e-03  5.68140585e-03
  2.59421686e-02  2.77671762e-03  5.68140585e-03  2.59421686e-02
  1.17946310e-02 -3.06074104e-02  2.45965822e-02  1.17946310e-02
 -3.06074104e-02  2.45965822e-02 -1.09008337e-02 -7.84783817e-03
  2.25468925e-02  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[-6.12357736e-03 -3.03655755e-01  1.22132090e-01 -6.12357736e-03
 -3.03655755e-01  1.22132090e-01 -6.12357736e-03 -3.03655755e-01
  1.22132090e-01 -2.85165088e-01  2.09674820e-01  1.73856532e-01
 -2.85165088e-01  2.09674820e-01  1.73856532e-01 -1.57864865e-01
  1.74028904e-01  1.26737569e-01 -1.57864865e-01  1.74028904e-01
  1.26737569e-01 -1.57864865e-01  1.74028904e-01  1.26737569e-01
 -1.13767758e-03  1.34692080e-01  8.90841344e-02 -1.13767758e-03
  1.34692080e-01  8.90841344e-02 -1.13767758e-03  1.34692080e-01
  8.90841344e-02  7.18151353e-02 -7.08814724e-02  1.15520796e-01
  7.18151353e-02 -7.08814724e-02  1.15520796e-01  6.60732339e-02
  6.95825371e-02  1.38538957e-01  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

1.4 3. Discriminator

1.5 3.1 Elementary Circuit Layer

The parameter "W" is a matrix containg the rotation angles applied to the circuit. It has "num_qubits" rows and three colums.

```
In [5]: def layer(num_wires,W):
            for i in range(num_wires):
               qml.Rot(W[i, 0], W[i, 1], W[i, 2], wires=i)
            for i in range(num_wires):
                qml.CNOT(wires=[i, (i+1)%num_wires])
```

1.5.1 3.2 Quantum node for real data

The first parameter "real_values" is an array that contains the n input real data points. The normalized input data is encoded in the amplitudes of "num_qubits". The second parameter "weights" is a matrix containing the rotation angles applied to the rotation gates. The matrix has one row per layer and "num_qubits" columns. The wire 0 is the output of the circuit. The output ranges in the continuous interval +1 down to -1, respectively corresponding to qubits $|0\rangle$ and $|1\rangle$. Intermediate values represent superpositions of qubits $|0\rangle$ and $|1\rangle$. The optimizer aims at finding angles such that the output of the circuit is approaching $+1$, which corresponds to qubit $|0\rangle$.

```
In [6]: @qml.qnode(dev)
        def real_disc_circuit(real_values,weights):
            # initialize the input state real data
            qml.QubitStateVector(real_values,wires=range(num_qubits))
            # assign weights to layers
            for W in weights:
                layer(num_qubits,W)
            # measure
            return qml.expval.PauliZ(0)
```

```
In [7]: @qml.qnode(dev)
        def gen_disc_circuit(fake_values,disc_weights):
            # initialize with generator data
            qml.QubitStateVector(fake_values, wires=range(num_qubits))
            # discriminator layers
            for W in disc_weights:
                layer(W)
            # measure
            return qml.expval.PauliZ(0)
```

1.5.2 3.3 Initial circuit values.

The discriminator circuit is initialized with random rotation angles contained in variable "disc_weights".

```
In [8]: np.random.seed(0)
        num_layers = 2
        # discriminator weights
        disc_weights = 0.01 * np.random.randn(num_layers, num_qubits, 3)
        # test the discriminator circuit
        r = real_disc_circuit(real_values[0,:],disc_weights)
        assert(r>=-1 and r<=1)
        print("Discriminator expectation (test mode): ", r)
```

```
Discriminator expectation (test mode):  -0.31866640473431623
```

1.5.3 3.4 Discriminator Optimization

Probability of correctly classifying real data The output of the discriminator r is a value in the continuous interval $+1$ down to -1, i.e., $|0\rangle$ and $|1\rangle$. The output is interpreted as follows. When the output is $+1$, the data is accepted as True. When it is -1, the data is rejected and considered fake. The output r is converted to a probabiliy value, in the interval $[0,1]$, using the following conversion:

$$p = \frac{r+1}{2}$$

Parameter "values" is an arry of n normalized data points. Parameter "disc_weights" is a matrix of angles used in the discriinator circuit.

```
In [ ]: def prob_real_true(real_values,disc_weights):
            # get measurement
            r = real_disc_circuit(real_values,disc_weights)
            assert(r>=-1 and r<=1)
            # convert "r" to a probability
            p = (r + 1) / 2
            assert(p>=0 and r<=1)
            return p
```

Discriminator cost function The discriminator aims to maximize the probability p_R of accepting true data while minimizing the probability p_F of accepting fake data. During the optimizaton of the discriminator, the optimizer, being gradient descent, tries to minimize the cost represented by the term $-p_R$.

```
In [ ]: def disc_cost(real_values,disc_weights):
            cost = - prob_real_true(real_values,disc_weights)
            return cost
```

Generate initial fake data

```
In [ ]: # random fake values
        fake_values =  0.01 * np.random.randn(2**num_qubits)
        # normalized fake values
        norm_fake_values = normalize(fake_values)
        # check consistency of probability amplitudes
```

```
        assert np.abs(1-np.sum(norm_fake_values ** 2))<=0.0000001
        print("Normalized fake data: ",norm_fake_values)
        # test the generator circuit
        #r = gen_disc_circuit(norm_fake_values,disc_weights)
        #assert(r>=-1 and r<=1)
        #print("Generator expectation (test mode): ", r)
```

```
Normalized fake data:  [ 0.16451982  0.16078746 -0.05179503 -0.04042524 -0.14021706 -0.1898909
 -0.22816986  0.26086616 -0.0681529  -0.0585812  -0.16752924  0.10396939
 -0.21581743 -0.02844855 -0.11974568  0.05173828 -0.06830708 -0.15787927
 -0.00376865  0.0572784   0.00889497  0.04044786 -0.08482431 -0.04850733
 -0.08992433 -0.04808101 -0.10873745 -0.23084601  0.02372619 -0.05372789
 -0.21799721  0.06188525 -0.12132788  0.00694636  0.09749716  0.01724816
  0.15236561 -0.16512627  0.05380287 -0.09157578 -0.11644678 -0.07740629
 -0.04166216  0.00751067 -0.1558089   0.12046243  0.06227041 -0.20543318
  0.19901555  0.25352654  0.15763153 -0.02406033 -0.14318569  0.14100587
 -0.05391457  0.16347067  0.02785144  0.13060042  0.04765486  0.09448603
  0.00140411  0.23881436  0.01697124  0.05375576]
```

Discriminator optimization The outcome of the optimization of the discriminator is a matrix of rotation angles (weights) actualizing the discriminator circuit such that the probability that real data is reconized as tru is maximized.

```
In [ ]: # create the optimizer
        opt = GradientDescentOptimizer(0.1)
        for i in range(50):
            j = 0
            disc_weights = opt.step(lambda v: disc_cost(real_values[j,:],v),disc_weights)
            cost = disc_cost(real_values[j,:],disc_weights)
            if i % 5 == 0:
                print("Step {}: cost = {}".format(i+1, cost))
        p_R = prob_real_true(real_values[0,:],disc_weights)
        assert(p_R>=0 and p_R<=1)
        print("Probability of real true: ", p_R)
        #p_F = prob_fake_true(norm_fake_values, disc_weights)
        #assert(p_F>=0 and p_F<=1)
        #print("Probability of fake true: ",p_F)
```

```
Step 1: cost = -0.3600021301305306
Step 6: cost = -0.48012090716650735
Step 11: cost = -0.6267257784058954
Step 16: cost = -0.7646215871267541
Step 21: cost = -0.8631753136938662
Step 26: cost = -0.9200803809217414
Step 31: cost = -0.9492141305426494
Step 36: cost = -0.9635297581772706
Step 41: cost = -0.9706756627495222
```

```
Step 46: cost = -0.9744598940355292
Probability of real true:  0.9762990269792956
```

1.6 4. Generator

The generator produces fake data with entropy, i.e., with uncertainty.

The fake data can be generated by a qubit-quantum circuit, but it is not usable for perpetrating attacks in the classical data world. Qubit-uantum circuits cannot generate continuous-domain classical data.

The generator is built using photonic quantum computing and a circuit containing only Gaussian operations that can generate data in a continuous domain.

A photonic quantum computing Gaussian device is created:

```
In [ ]: dev_gaussian = qml.device('default.gaussian', wires=2**num_qubits)
```

1.6.1 4.1 Construction of Photonic Quantum Node

The input on each wire is the vaucuum state $|0>$, i.e., no photon on the wire. The first gate is a displacement gate, with parameter α, that phase shift the qumod. The parameter α is a psecified in the polar form, as a magnitude (mag_alpha) and a an angle (phase_alpha). This is an active transformation that modifies the photonoc energy of the system.

The second gate rotate the qumode by an angle ϕ. The measured mean number of photon is $< \hat{a}\dagger\hat{a} >$, i.e., the average number of photons in the final state.

```
In [ ]: #@qml.qnode(dev_gaussian)
        #def mean_photon_gaussian(params):
        #    qml.Displacement(params[0],params[1], wires=0)
        #    qml.Rotation(params[2], wires=0)
        #    return qml.expval.MeanPhoton(0)
```

```
In [ ]: @qml.qnode(dev_gaussian)
        def mean_photon_gaussian(params):
            for i in range(2**num_qubits):
                qml.Displacement(params[i][0],params[i][1], wires=i)
                qml.Rotation(params[i][2], wires=i)
            #qml.Displacement(params[0][0],params[0][1], wires=1)
            #qml.Rotation(params[0][2], wires=1)
            return [qml.expval.MeanPhoton(i) for i in range(2**num_qubits)]
```

Using arbitrary displacement and rotatio verify the generator circuit:

```
In [ ]: init_params = 0.1*np.ones([2**num_qubits,3],dtype=float)
        mean_photon_gaussian(init_params)
```

```
Out[ ]: array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
               0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
               0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
               0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
               0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,
               0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01])
```

1.6.2 4.2 Generator Cost Function

Using a Gaussian device, displacement and rotation angle pairs (in "params") are applied successively to the photonic quantum node. The expected numbers of output photons are measured for each case. Measurement results are stored into variable "fake_values". There is one fake value per probability amplitude, i.e., 2^6.

The fake values are normalized and applied to the discriminator circuit, using the rotation angles determined during its optimization.

```
In [ ]: def gen_cost(params,disc_weights):
            # calculate expected number of photons
            fake_values = mean_photon_gaussian(params)
            # normalize fake values
            norm_fake_values = normalize(fake_values)
            assert np.abs(1-np.sum(norm_fake_values ** 2))<=0.0000001
            # determine the probability of recognizing them as true values
            cost = - prob_real_true(fake_values,disc_weights)
            return cost
```

Verify cost function:

```
In [ ]: # generator weights
        gen_weights = 0.1*np.ones([2**num_qubits,3],dtype=float)
        gen_cost(gen_weights,disc_weights)

Out[ ]: -0.4999946815954298
```

1.6.3 Generator optimization

Perform gradient descent optimization iterations:

```
In [ ]: # initialize the optimizer
        opt = qml.GradientDescentOptimizer(stepsize=0.1)
        # set the number of steps
        steps = 20
        print("...")
        for i in range(steps):
            # update the circuit parameters
            gen_weights = opt.step(lambda v: gen_cost(v,disc_weights),gen_weights)
            cost = gen_cost(gen_weights,disc_weights)
            if i % 5 == 0:
                print("Step {}: cost = {}".format(i+1, cost))

...
Step 1: cost = -0.5000202786224744


In [ ]:
```