Project_3

November 20, 2024

1 Introduction to Quantum Information and Quantum Machine Learning

Instructor: Dr Sci. Eng. Przemysław Głowacki

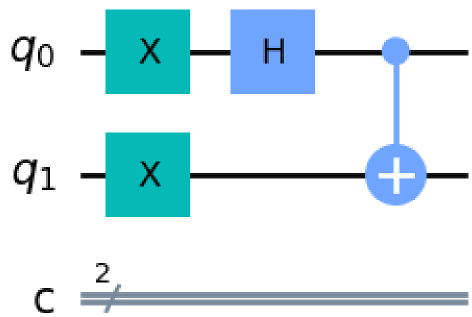
Kacper Dobek 148247

[301]:
```
from qiskit import (
    QuantumRegister,
    ClassicalRegister,
    QuantumCircuit,
    execute,
    Aer,
    BasicAer,
    IBMQ,
)
from qiskit.compiler import transpile, assemble
from qiskit.visualization import (
    plot_state_city,
    plot_bloch_multivector,
    plot_state_hinton,
    plot_state_qsphere,
    plot_histogram,
    plot_distribution,
)
from numpy import pi
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from collections import Counter
from IPython.display import display, Markdown
```

[302]:
```
N_REPS = 1024

# selection of quantum simulator (or processor)
QASM_BACKEND = Aer.get_backend("qasm_simulator")
```

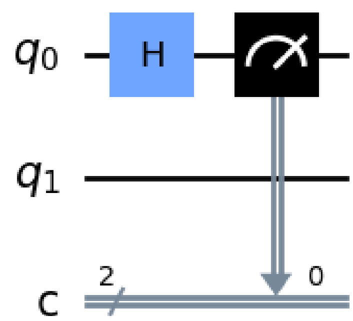
1.0.1 Create Charlie's circuit

[303]:
```
qreg_q = QuantumRegister(2, "q")
creg_c = ClassicalRegister(2, "c")
circuit_Charlie = QuantumCircuit(qreg_q, creg_c)

circuit_Charlie.x(qreg_q[0])
circuit_Charlie.x(qreg_q[1])
circuit_Charlie.h(qreg_q[0])
circuit_Charlie.cx(qreg_q[0], qreg_q[1])

display(circuit_Charlie.draw(output="mpl"))
```

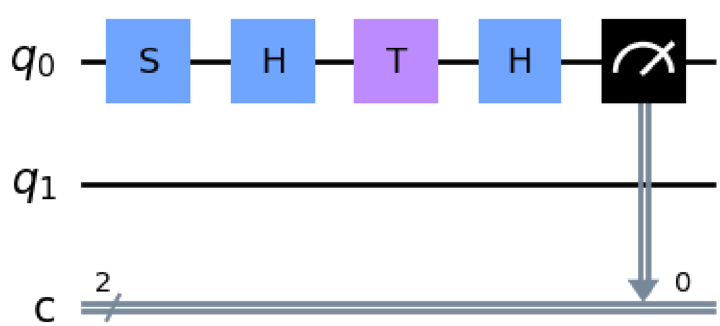


1.0.2 Alice's circuits

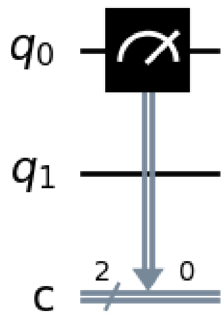
[304]:
```
circuit_Alice_X = QuantumCircuit(qreg_q, creg_c)
circuit_Alice_X.h(qreg_q[0])
circuit_Alice_X.measure(qreg_q[0], creg_c[0])
display(circuit_Alice_X.draw(output="mpl"))
```



[305]:
```
circuit_Alice_W = QuantumCircuit(qreg_q, creg_c)
circuit_Alice_W.s(qreg_q[0])
circuit_Alice_W.h(qreg_q[0])
circuit_Alice_W.t(qreg_q[0])
circuit_Alice_W.h(qreg_q[0])
circuit_Alice_W.measure(qreg_q[0], creg_c[0])

display(circuit_Alice_W.draw(output="mpl"))
```

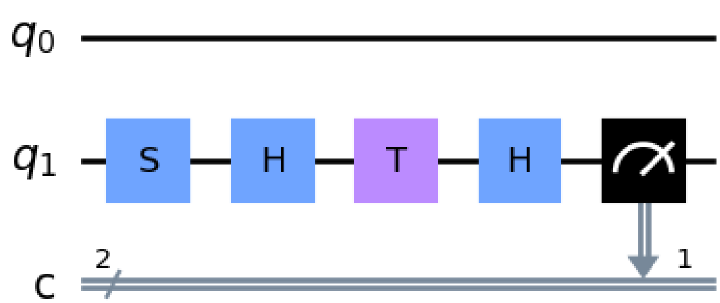


[306]:
```
circuit_Alice_Z = QuantumCircuit(qreg_q, creg_c)
circuit_Alice_Z.measure(qreg_q[0], creg_c[0])

display(circuit_Alice_Z.draw(output="mpl"))
```

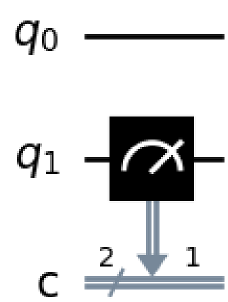


1.0.3 Bob's circuits

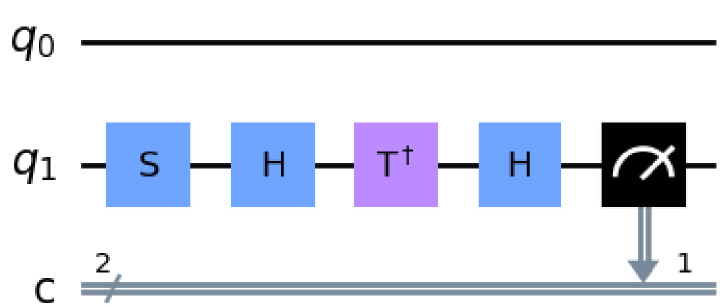
[307]:
```
circuit_Bob_W = QuantumCircuit(qreg_q, creg_c)
circuit_Bob_W.s(qreg_q[1])
circuit_Bob_W.h(qreg_q[1])
circuit_Bob_W.t(qreg_q[1])
circuit_Bob_W.h(qreg_q[1])
circuit_Bob_W.measure(qreg_q[1], creg_c[1])

display(circuit_Bob_W.draw(output="mpl"))
```



[308]:
```
circuit_Bob_Z = QuantumCircuit(qreg_q, creg_c)
circuit_Bob_Z.measure(qreg_q[1], creg_c[1])

display(circuit_Bob_Z.draw(output="mpl"))
```



[309]:
```
circuit_Bob_V = QuantumCircuit(qreg_q, creg_c)
circuit_Bob_V.s(qreg_q[1])
circuit_Bob_V.h(qreg_q[1])
circuit_Bob_V.tdg(qreg_q[1])
circuit_Bob_V.h(qreg_q[1])
circuit_Bob_V.measure(qreg_q[1], creg_c[1])

display(circuit_Bob_V.draw(output="mpl"))
```



[310]:
```
# Put circuits in dictionaries
alice_circuits = {"X": circuit_Alice_X, "W": circuit_Alice_W, "Z":␣
 ↪circuit_Alice_Z}
bob_circuits = {"W": circuit_Bob_W, "Z": circuit_Bob_Z, "V": circuit_Bob_V}

# Randomly select the measurement type for Alice and Bob
```

```
alice_mesurement_types = np.random.choice(["X", "W", "Z"], N_REPS)
bob_mesurement_types = np.random.choice(["W", "Z", "V"], N_REPS)
```

[311]:
```
# Define a dictionary to store relevant results

results = {
    "XW": [],
    "XV": [],
    "ZW": [],
    "ZV": [],
}
```

[312]:
```
# Combine the circuits and run experiments
for i in range(N_REPS):
    # Combine the circuits
    circuit = circuit_Charlie.compose(
        alice_circuits[alice_mesurement_types[i]]
    ).compose(bob_circuits[bob_mesurement_types[i]])

    # Execute the circuit
    job_sim = execute(circuit, QASM_BACKEND, shots=1)
    result = job_sim.result().get_counts(circuit)
    result = list(result.keys())[0]

    # Store the result
    measurement_type = alice_mesurement_types[i] + bob_mesurement_types[i]
    if measurement_type in results.keys():
        results[measurement_type].append(result)
```

1.0.4 Process the results

[313]:
```
table = pd.DataFrame()

for key in results.keys():
    row = pd.DataFrame({"count": Counter(results[key])})
    row["original_a"] = row.index.map(lambda x: x[0])
    row["original_ap"] = row.index.map(lambda x: x[1])
    row["a"] = row["original_a"].map(lambda x: 1 if x == "1" else -1)
    row["ap"] = row["original_ap"].map(lambda x: 1 if x == "1" else -1)
    row.drop(columns=["original_a", "original_ap"], inplace=True)
    row["N"] = row["count"].sum()
    row["measurement_type"] = (
        r"$\hat{" + str(key[0]) + "} \otimes \hat{" + str(key[1]) + "}$"
    )
    row["freq"] = row["count"] / row["N"]
    row["prob"] = row["a"] * row["ap"] * row["freq"]
    row["prob_sum"] = row["prob"].sum()
```

```
    row.set_index(["measurement_type", "prob_sum", "N", "a"], inplace=True)
    table = pd.concat([table, row])
```

Calculate S value

[314]:
```
prob_sum = table.index.get_level_values("prob_sum").unique()
S = prob_sum[0] - prob_sum[1] + prob_sum[2] + prob_sum[3]
```

[315]:
```
table.index.rename(
    [
        "Measurement type",
        r"$\langle \hat{A} \otimes \hat{B} \rangle = \sum_{a, a'}␣
↪p_{jk}(a,a')\times(a \times a')$",
        r"$N_{jk}$",
        "a",
    ],
    inplace=True,
)
```

[316]:
```
# Reorder the columns
table = table[["ap", "count", "freq", "prob"]]
```

[317]:
```
# Rename the columns to use symbols
table.rename(
    columns={
        "ap": r"$a'$",
        "count": r"$n_{jk}(a,a')$",
        "freq": r"$p_{jk}(a,a')$",
        "prob": r"$p_{jk}(a,a') \times (a \times a')$",
    },
    inplace=True,
)
```

[318]:
```
# Due to latex rendering issues, the table is pasted as an image

# display(Markdown(table.to_latex()))
```

Measurement type	$\langle\hat{A}\otimes\hat{B}\rangle = \sum_{a,a'} p_{jk}(a,a') \times (a \times a')$	N_{jk}	a	a'	$n_{jk}(a,a')$	$p_{jk}(a,a')$	$p_{jk}(a,a') \times (a \times a')$
$\hat{X}\otimes\hat{W}$	-0.633028	109	1	-1	55	0.504587	-0.504587
			-1	-1	8	0.073394	0.073394
			-1	1	34	0.311927	-0.311927
			1	1	12	0.110092	0.110092
$\hat{X}\otimes\hat{V}$	0.811966	117	1	1	56	0.478632	0.478632
			1	-1	6	0.051282	-0.051282
			-1	-1	50	0.427350	0.427350
			-1	1	5	0.042735	-0.042735
$\hat{Z}\otimes\hat{W}$	-0.614035	114	1	-1	47	0.412281	-0.412281
			-1	1	45	0.394737	-0.394737
			1	1	9	0.078947	0.078947
			-1	-1	13	0.114035	0.114035
$\hat{Z}\otimes\hat{V}$	-0.792000	125	1	-1	49	0.392000	-0.392000
			-1	1	63	0.504000	-0.504000
			1	1	5	0.040000	0.040000
			-1	-1	8	0.064000	0.064000

[319]:
```
print(f"S = {S}")
```

```
S = -2.8510284226208906
```