A practical use case for Quantum Generative Adversarial Networks in High Energy Physics

Generating top squark events decaying via the four-body mode in single-lepton final states in proton-proton collisions at $\sqrt{s} = 13$ TeV

Diogo Carlos Chasqueira de Bastos

Advanced Topics in Particle and Astroparticle Physics II

Abstract

This is a simple paragraph at the beginning of the document. A brief introduction about the main subject.

Contents

List of Figures

List of Tables

Glossary

CERN QTI CERN Quantum Technology Initiative. 1

CMS Compact Muon Solenoid. 1, 14

HEP High Energy Physics. 1–3

LSP Lightest Supersymmetric Particle. 13

MC Monte Carlo. 1, 14, 15

NISQ Noisy Intermediate-Scale Quantum. 1

QC Quantum Computing. 1–3, 14

qGAN quantum Generative Adversarial Network. ii, 1, 3, 11, 12, 15, 19, 24

QML Quantum Machine Learning. 1, 2, 6

qNN quantum Neural Network. 11, 21

SM Standard Model. 13

SUSY Supersymmetry. 13

1 Introduction

One of the main objectives of CERN Quantum Technology Initiative (CERN QTI) is to investigate if Quantum Computing (QC) can be used in the field of High Energy Physics (HEP) in the Noisy Intermediate-Scale Quantum (NISQ) era. Noisy means that, currently, we have imperfect control over qubits. Intermediate refers to the number of qubits our present quantum computers have. They range from 50 to a few hundred. In order to fully fulfill the promise of quantum computing, the challenges of noise and scalability (millions of qubits) need to be solved. Nonetheless, we can already use the available quantum computers and algorithms to tackle present challenges.

Quantum Machine Learning (QML) is a growing research area that explores the interplay of ideas from QC and machine learning. This project explores the use of qGANs [19], a QML algorithm, to learn the kinematic distributions of the top squark (or stop) four-body decays [16] in Compact Muon Solenoid (CMS) data. The Feynman diagram for such process is represented in Figure 1. In HEP to simulate such distributions, Monte Carlo (MC) simulations are used in a very convoluted and computational resources hungry process that can take up months. As we will see, the method presented in this project could speed up this process significantly when run on quantum computers. This method could also be used for data augmentation of the MC generated samples which would be helpful in the training of the classification algorithms in many HEP searches improving the sensitivity of such searches.

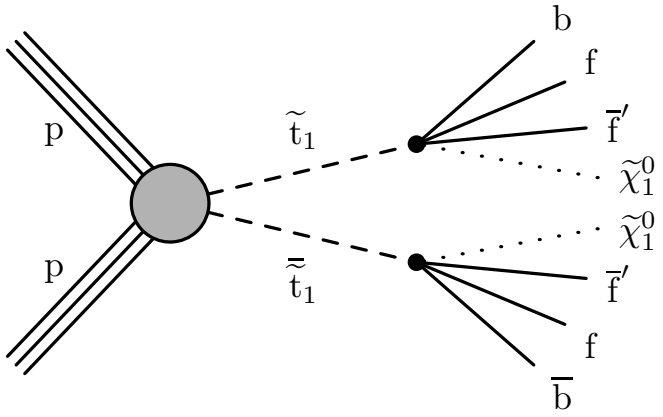

Figure 1: Diagram of top squark pair production $\widetilde{t}_1\bar{\widetilde{t}}_1$ in pp collisions, with a four-body decay of each top squark.

This project is implemented in python [17] and will be run on a classical computer (my personal laptop) simulating a quantum computer with five qubits using pennylane [14] to train and optimize the quantum algorithm, and cirq [7] for writing, manipulating, and running the quantum simulator. Since QML is a novel technique in HEP, after the present Section 1, a brief introduction to QC and the necessary operations needed for the final algorithm are presented in Section 2. The main development of this project is, as well as its implementation, shown in Section 3. The final results of the project and a brief discussion of futures steps to be built on top of this project are reserved to Section 3.8.

2 A Brief Introduction to Quantum Computing

In order to develop a qGAN and apply it in the context of HEP it is important to first go over the QC concepts necessary to design this algorithm. Section 2 focus on that.

2.1 Meet the Qubit

A quantum bit, know as a qubit, is the basic unit of quantum information. The principle behind QC is to take advantage of the properties of quantum mechanics for computations and information processing such as superposition and entanglement. A classic bit can only have a value of 0 or 1, a qubit can be in a superposition of 0 and 1. The Bloch sphere [4] is a geometrical representation of the pure state space of a qubit as is shown in Figure 2. This is a useful representation of a qubit where one can think of the sate of a qubit as a vector inside a 3-D sphere. The north and south poles of the Bloch sphere are typically chosen to correspond to the standard basis vectors $|0\rangle$ and $|1\rangle$ respectively.

The qubit's 0 and 1 states are mathematically represented as follows:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{1}$$

As a consequence of this mathematical description, unlike a bit, a qubit is not limited to being in one of these two states. Qubits can exist in what's known as a superposition of states. A superposition is a linear combination of two basis vectors.

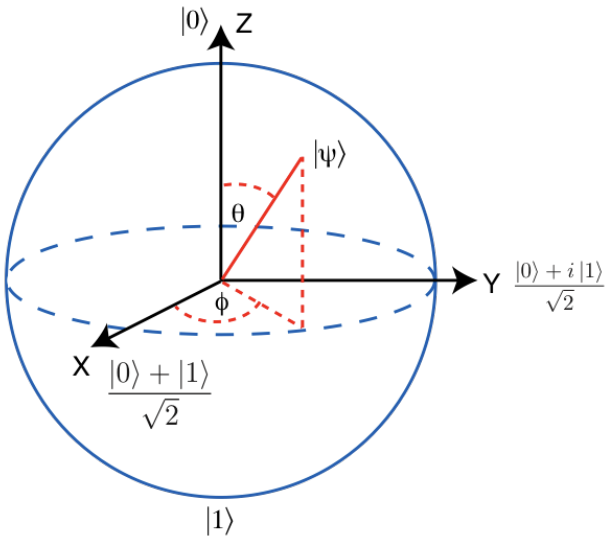

Figure 2: Representation of the qubit in the state $|\psi\rangle$ in the Bloch sphere.

Mathematically:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2}$$

where α and β are complex numbers that satisfy:

$$\alpha\alpha^* + \beta\beta^* = 1 \tag{3}$$

2.2 Quantum Circuits

Quantum circuits are a common means of visually representing the sequence of operations that are performed on qubits during a quantum computation. They consist of a set of operations (gates) applied to a set of qubits (wires). Each wire in the diagram represents a qubit. Circuits are read from left to right, and this is the order in which operations are applied. An example of a quantum circuit is shown in Figure 3.

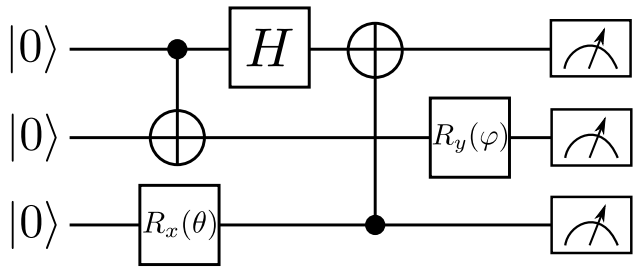

Figure 3: Example of a quantum circuit. This circuit has three qubits that start in the state $|0\rangle$, performs five operations, and measures every qubit at the end of the circuit.

A quantum computation is all about manipulating qubit states by using gates and seeing the outcome of such manipulations by measuring the qubits.

2.3 Flipping qubits

Performing quantum operations involves multiplication by matrices that send valid, normalized quantum states to other normalized quantum states. The simplest quantum operation as the following effect:

$$X|0\rangle = |1\rangle,\ X|1\rangle = |0\rangle \tag{4}$$

Known as bit flip, $Pauli{-}\mathrm{X}$ or NOT gate, it can be mathematically represented as:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{5}$$

Figure 4 represents the qubit flip gate in a quantum circuit diagram.

Equation 6 shows the mathematical operation of applying the

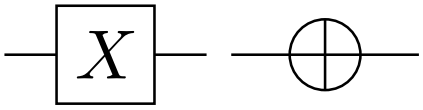

Figure 4: The *Pauli*−X or *NOT* gate in a quantum circuit.

NOT gate to a qubit in state $|0\rangle$ flipping it to state $|1\rangle$.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{6}$$

2.4 Rotations, rotations, rotations: the RX, RY, and RZ Gates

One of the most relevant operations to the qubit in the context QML are the $R_X(\theta)$, $R_Y(\theta)$, and $R_Z(\theta)$ gates because they can be used to rotate the qubit state in the Bloch sphere. This means qubits can be manipulated as a function of a parameter, an angle, in order to find an optimum in the quantum algorithm being designed. This concept is analogous to using weights in Neural Networks. In the same manner, the value of the angles used in the rotations are the parameter upon which the QML algorithm is going to be trained.

As it was shown, a qubit can be represented in 3D-space by the Bloch sphere. Using the same representation, $R_X(\theta)$, $R_Y(\theta)$, and $R_Z(\theta)$ are shown in Figure 5.

The matrix representation of these rotations:

$$R_X(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} \tag{7}$$

$$R_Y(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} \tag{8}$$

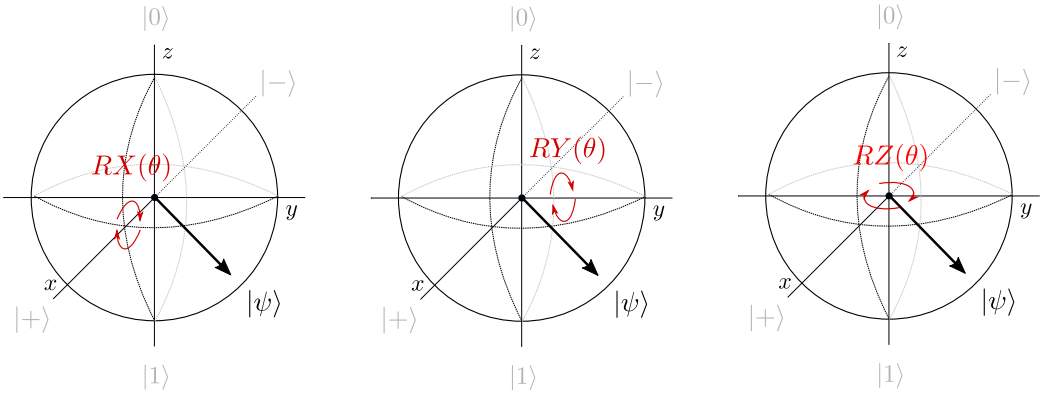

Figure 5: The $R_X(\theta)$, $R_Y(\theta)$, and $R_Z(\theta)$ gates in a the Bloch sphere.

$$R_Z(\theta) = \begin{pmatrix} e^{\frac{-i\theta}{2}} & 0 \\ 0 & e^{\frac{i\theta}{2}} \end{pmatrix} \tag{9}$$

Figure 6 represents the $R_X(\theta)$, $R_Y(\theta)$, and $R_Z(\theta)$ gates in a quantum circuit diagram.

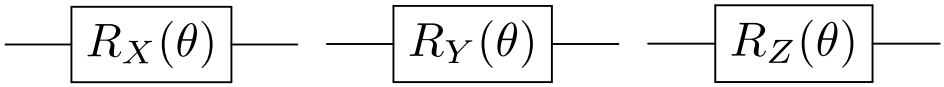

Figure 6: The $R_X(\theta)$, $R_Y(\theta)$, and $R_Z(\theta)$ gates in a quantum circuit diagram.

2.5 Superposition and the Hadamard Gate

The Hadamard gate, H, is a fundamental quantum gate that allows to move away from the poles of the Bloch sphere, and create a superposition of $|0\rangle$ and $|1\rangle$ as follows:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{10}$$

It is mathematically represented as:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{11}$$

Figure 7 represents the *Hadamard* gate in a quantum circuit diagram.

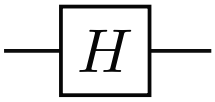

Figure 7: The H gate in a quantum circuit diagram.

Equation 12 shows the mathematical operation of applying the H gate to a qubit in state $|0\rangle$ mapping it to a state in a superposition, commonly written as $|+\rangle$.

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle \tag{12}$$

2.6 Multiple Qubit States

To describe a state of 2 qubits, 4 complex amplitudes are required:

$$|\psi\rangle = \psi_{00}|00\rangle + \psi_{01}|01\rangle + \psi_{10}|10\rangle + \psi_{11}|11\rangle = \begin{bmatrix} \psi_{00} \\ \psi_{01} \\ \psi_{10} \\ \psi_{11} \end{bmatrix} \tag{13}$$

Such that the normalization condition is respected:

$$|\psi_{00}|^2 + |\psi_{01}|^2 + |\psi_{10}|^2 + |\psi_{11}|^2 = 1 \tag{14}$$

Qubits $|a\rangle$ and $|b\rangle$ are two separate qubits such that:

$$|a\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}, |b\rangle = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \tag{15}$$

The kronecker product is used to describe the collective state. The collective state of qubits $|a\rangle$ and $|b\rangle$ is described as follows:

$$|ba\rangle = |b\rangle \otimes |a\rangle = \begin{bmatrix} b_0 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \\ b_1 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} b_0 a_0 \\ b_0 a_1 \\ b_1 a_0 \\ b_1 a_1 \end{bmatrix} \tag{16}$$

$$|ba\rangle = b_0 a_0 |00\rangle + b_0 a_1 |01\rangle + b_1 a_0 |10\rangle + b_1 a_1 |11\rangle \tag{17}$$

For a computation with n qubits, a computer needs to keep track of 2^n complex amplitudes. This is the reason why quantum computers are difficult to simulate. A modern laptop can simulate a general quantum state of around 10 qubits, but simulating 100 qubits is too difficult even for the largest supercomputers.

2.7 Entanglement and the CNOT Gate

Until now, only gates that act on a single qubit have been described. In order to complete all the "ingredients" necessary, gates that act on multiple qubits are needed in order to entangle multiple qubits.

When 2 qubits are entangled, by knowing the state of one qubit, the state of other becomes known as well. The $CNOT$ operation changes the state of a target qubit depending on the state of a control qubit. When these two concepts are merged, qubits can be entangled in quantum computers.

By definition, a state is entangled if it cannot be described as a tensor product of individual qubit states. An entangled state can only be described by specifying the full state. One example of an entangled state:

$$|ba\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{18}$$

In order to describe this state using the kronecker tensor one needs to find a_0, a_1, b_0 and b_0 such that:

$$b_0a_0 = 1, b_0a_1 = 0, b_1a_0 = 0, b_1a_1 = 1 \tag{19}$$

Each variable appears in two equations, one of which is equal to 1, and the other to 0. But for any of the 0 ones to be true, at least one variable needs to be 0, and that would immediately contradict the other equation. Therefore, there is no solution, it's not possible to describe this state as two separate qubits. They are entangled.

The *controlled*−NOT or $CNOT$ gate can make an entangled state by performing the $Pauli$−X operation on one target qubit depending on the state of another control qubit. Mathematically represented as:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{20}$$

Figure 8 represents the $CNOT$ gate in a quantum circuit diagram.

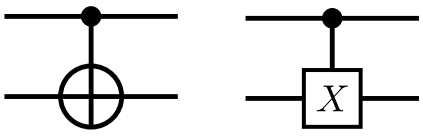

Figure 8: The *controlled*−NOT or $CNOT$ gate in a quantum circuit.

Qubits $|q_0\rangle$ and $|q_1\rangle$ are two separate qubits such that:

$$|q_0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |q_1\rangle = |0\rangle \tag{21}$$

The collective state is defined as $|q_0q_1\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$ which is not entangled. Now, if the gate $CNOT$ is applied to the collective state as in Equation 22, an entangled state is achieved.

$$CNOT|q_0q_1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{22}$$

$$CNOT|q_0q_1\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{23}$$

2.8 Quantum Generative Adversarial Network

qGANs are fundamentally composed of two quantum Neural Networks (qNNs), a generator and a discriminator, that compete between each other. The generative network generates candidates while the discriminating network evaluates them. The contest operates in terms of data distributions. Typically, the generative network learns to map from a latent space to a data

distribution of interest, while the discriminating network distinguishes candidates produced by the generator from the true data distribution. The generative network training objective is to increase the error rate of the discriminative network (i.e., "fool" the discriminator network by producing novel candidates that the discriminator thinks are part of the true data distribution). Figure 9 represents this idea.

This project will focus on the technical implementation details of a qGAN that can generate a 2-qubit state.

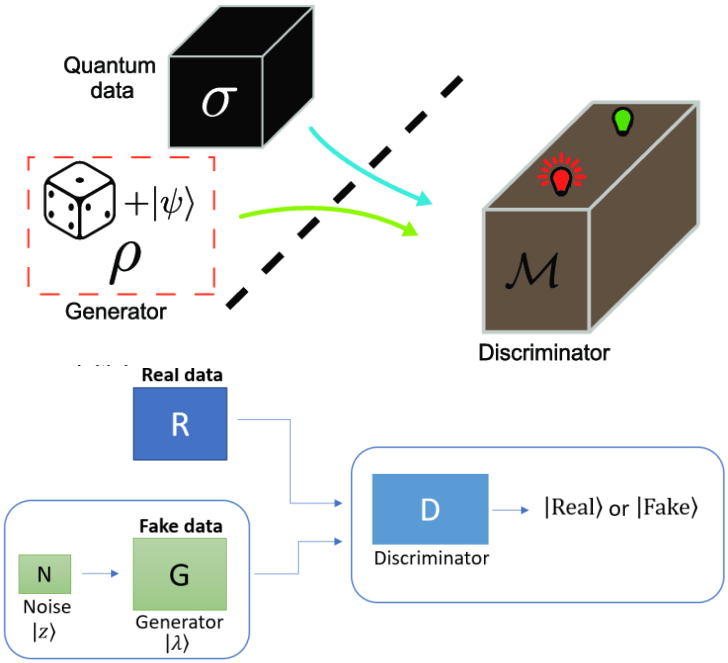

Figure 9: Representation of a generic qGAN.

3 Generating Stop Four-Body Decay Events

Supersymmetry (SUSY) [12, 18, 13, 11, 3, 9] predicts the existence of a scalar partner for each left-handed and right-handed fermion of the Standard Model (SM). Searches for SUSY are among the important focal points of the physics program at the CERN LHC, since SUSY naturally solves the problem of quadratically divergent loop corrections to the mass of the Higgs boson [1, 5, 6]. If R parity [10] is conserved, supersymmetric particles would be produced in pairs, and their decay chains would end with the Lightest Supersymmetric Particle (LSP), often considered to be the lightest neutralino $\widetilde{\chi}_1^0$. Such an LSP, being neutral, weakly interacting, and massive, would have the required characteristics for a dark matter particle, and thus, would offer a solution to another shortcoming of the SM. When the symmetry is broken, the scalar partners of an SM fermion acquire a mass different from the mass of the SM partner, with the mass splitting between scalar mass eigenstates being proportional to the mass of the SM fermion. Since the top quark is the heaviest fermion of the SM, the splitting between its chiral supersymmetric partners can be the largest among all supersymmetric quarks (squarks). The lighter supersymmetric scalar partner of the top quark, the top squark ($\widetilde{t}_1$), could therefore be the lightest squark. If SUSY is realized in nature, cosmological observations imply that the lightest top squark is almost degenerate with the LSP [2]. In this scenario, because the mass difference between the $\widetilde{t}_1$ and the $\widetilde{\chi}_1^0$ is smaller than the mass of the W boson, the two- and three-body decays of the $\widetilde{t}_1$ are kinematically forbidden, while the two-body decay to $c\widetilde{\chi}_1^0$ can be suppressed depending on the parameters of the model. This motivates the search for the four-body decay $\widetilde{t}_1 \to b\mathrm{f}\bar{\mathrm{f}}'\widetilde{\chi}_1^0$, where

b stands for the bottom quark, and the fermions f and $\bar{\mathrm{f}}'$ can be either quarks or leptons.

In this project a final state is considered, where the fermions f and $\bar{\mathrm{f}}'$ represent a charged lepton and its neutrino for the decay products of one $\widetilde{t}_1$, and two quarks for the other top squark. The considered final states contain at least one jet, a large missing transverse momentum, and exactly one charged lepton, which can be either an electron or a muon. The choice of final states where one top squark decays into a lepton is motivated by the decrease of the contributions from the multijet background in this mode, while increasing the selection efficiency with the other top squark decaying hadronically.The selected jet, attributed to initial-state radiation of a parton, is required to have high transverse momentum (p_{T}). Both neutralinos and the neutrino escape undetected, leaving high missing transverse momentum. Electrons and muons can be efficiently reconstructed and identified with p_{T} as low as 5.0 and 3.5 GeV, respectively.

Searching for this type of events in CMS data was the focus of my PhD research where the final signal selection was based on Boosted Decision Trees trained on MC simulated samples for signal and background. I noticed that one of the limitation to the classification algorithm was the statistical limit of the training samples. One technique that could tackle this is called data augmentation. In data analysis, data augmentation are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. This project explores the use of such a technique in a novel way by using QC. Signal simulated samples are encoded into quantum data in order to be processed by a simulated quantum computer. Then, from the signal kine-

matic distributions, the qGAN learns them and generates a new event with the same characteristics of the quantum data. After this, the generated event is transformed into a signal type event. Thus, generating new synthetic data.

The necessary code to run this project is open-source and can be found here: `https://github.com/diogodebastos/StopQGAN/blob/main/QGAN2qu.ipynb`.

3.1 Preparing Data

First, the stop signal MC sample is loaded. In order to select events in the kinematic region of interest a selection is applied. The selection is constituted by the following criteria:

$$p_{\mathrm{T}}(\ell) < 30 \text{ GeV} \;\&\&\; p_{\mathrm{T}}^{\mathrm{miss}} > 280 \text{ GeV} \;\&\&\; H_{\mathrm{T}} > 200 \text{ GeV} \;\&\&\; p_{\mathrm{T}}(\mathrm{ISR}) > 110 \text{ GeV} \tag{24}$$

where, $p_{\mathrm{T}}(\ell)$ is the lepto p_{T}, $p_{\mathrm{T}}^{\mathrm{miss}}$ is the missing transverse momentum, H_{T} is defined as the scalar p_{T} sum of all jets in the event, and p_{T}(ISR)the highest jet p_{T} attributed to initial-state radiation of a parton.

For the next step, the kinematic distributions to be generated are chosen. The number of variables chosen is four. This number is limited to the fact that this algorithm is run on a simulated quantum computer. Ideally, if more qubits were available, one would like to choose as many distributions as possible. The chosen kinematic variables are: the lepton p_{T} ($p_{\mathrm{T}}(\ell)$), the lepton charge (Charge(l)), the lepton η ($\eta(l)$), and the missing transverse momentum ($p_{\mathrm{T}}^{\mathrm{miss}}$). This distributions are shown in Figure 10.

Now that the kinematic variables have been chosen, they have

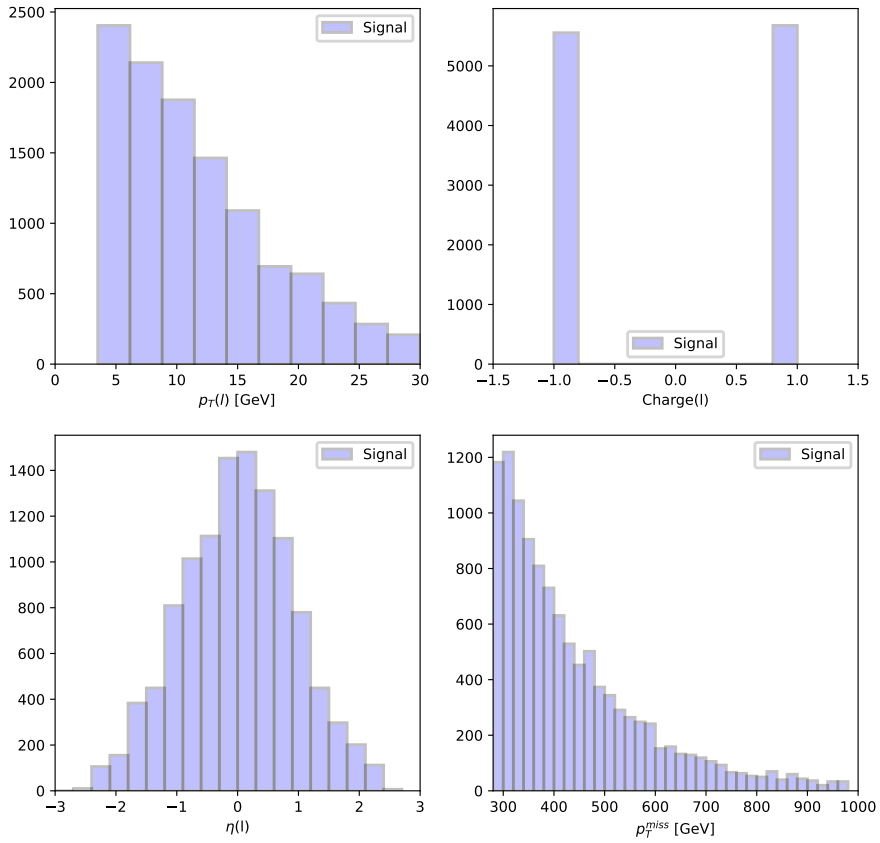

Figure 10: Distributions of $p_{\mathrm{T}}(\ell)$ (upper left), Charge(l) (upper right), $\eta(l)$ (lower left), and $p_{\mathrm{T}}^{\mathrm{miss}}$ (lower right) after the selection 24.

to be transformed into angles so that later, a qubit is created in a state that translates to the real value of the distribution. The $p_{\mathrm{T}}(\ell)$ and $p_{\mathrm{T}}^{\mathrm{miss}}$ are transformed into η angles, and Charge(l) and η(l) are transformed into ϕ angles according to Figure 2. The transformed variables are shown in Figure 11.

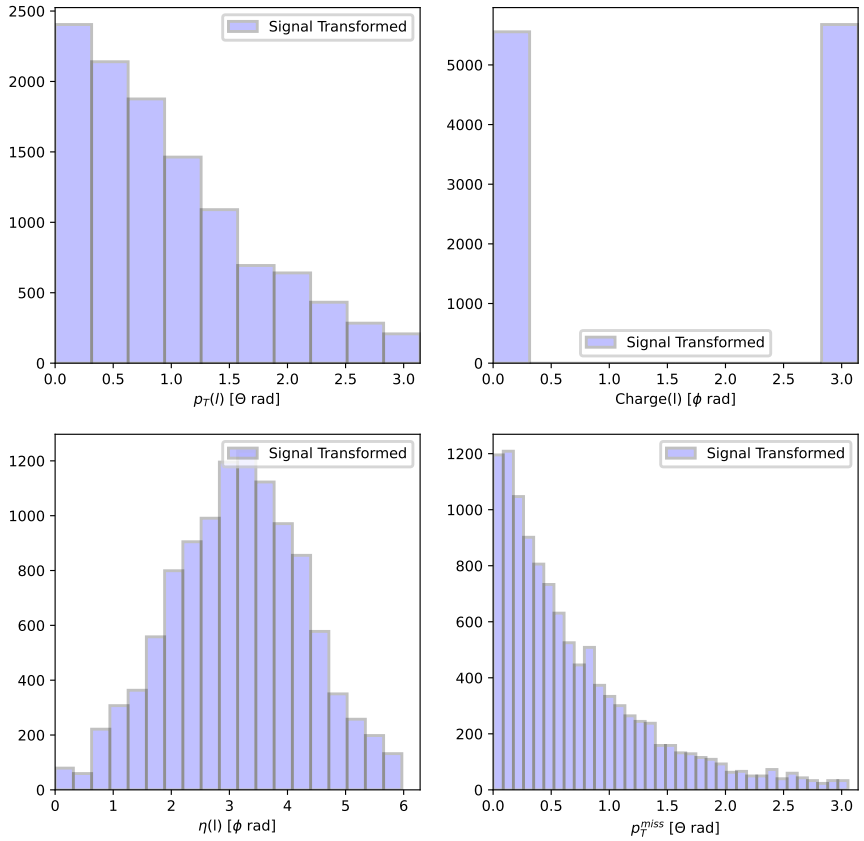

Figure 11: Distributions of $p_{\mathrm{T}}(\ell)$ (upper left), Charge(l) (upper right), $\eta(l)$ (lower left), and $p_{\mathrm{T}}^{\mathrm{miss}}$ (lower right) after the transformation.

Finally, all the events in the signal samples are translated into state vectors of two qubits. Qubit 0 is prepared to encode the $p_{\mathrm{T}}(\ell)$ and Charge(l) of a given event and qubit 1 is prepared to encode the $p_{\mathrm{T}}^{\mathrm{miss}}$ and η(l). Figure 12 helps to visualize this.

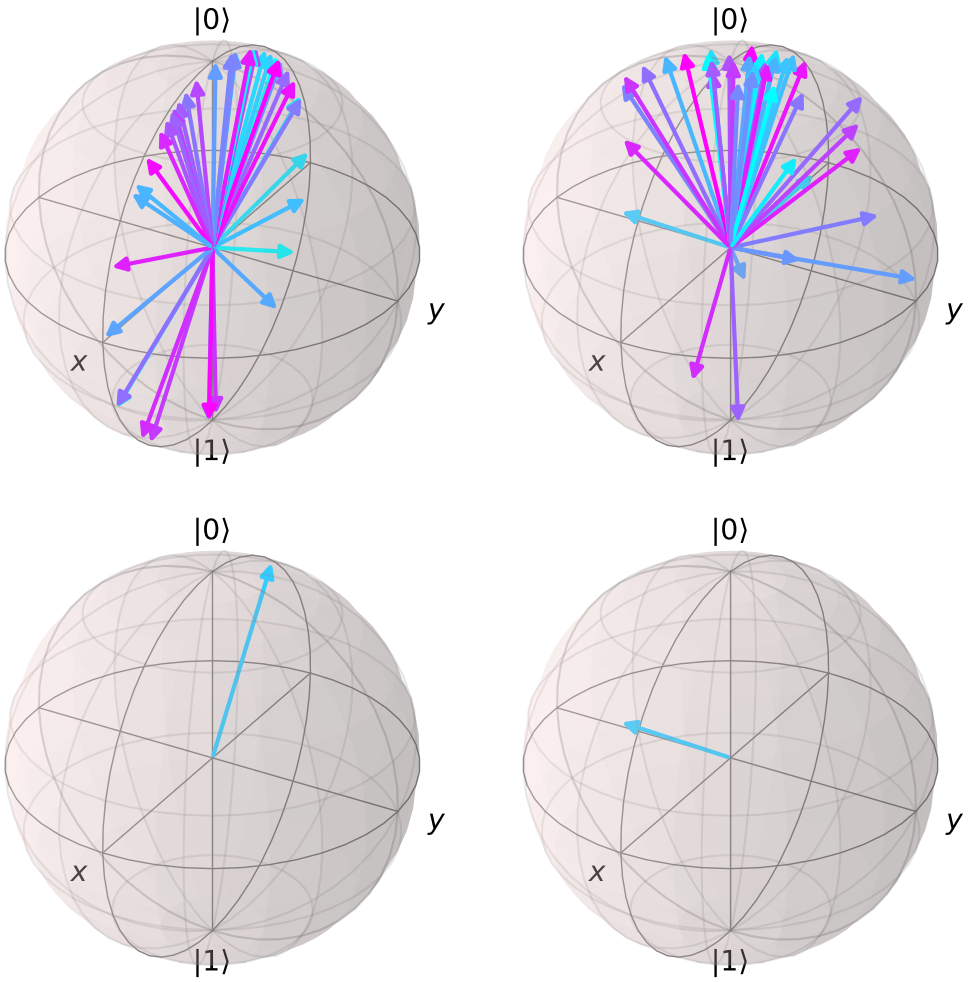

Figure 12: On top: all the possible states of qubit 0 and qubit 1. On the bottom, the example of a single signal event in qubit 0 and qubit2.

Another limitation to running this project in a simulated quantum computer, is the number of events that can be trained

on and generated. From now on, this project focus on learning the collective state of qubit 0 and qubit 1 of a single event and generating a synthetic state of two qubits that could represent the initial signal sample. Ideally, this algorithm would be deployed on a quantum computer for every event. This will be discussed in Section 3.8.

3.2 Designing a qGAN

Given that this section is focused on the design of the qGAN, python code snippets will be shared for a comprehensive and complete description of the necessary tools to reproduce the project results.

The first step is to load the necessary python libraries:

```
import pennylane as qml
import tensorflow as tf
```

A 5-qubit simulator device running in Cirq is used to design the qGAN. Each of the qubits has a role in the algorithm. The roles are:

- Qubits **0** and **1**: the 2-qubit state that we are trying to generate
- Qubits **2** and **3**: the generator's playground
- Qubit **4**: the discriminator's guess

The 5-qubit simulator quantum device is created by doing the following:

```
dev = qml.device('cirq.simulator', wires=5)
```

The *real* function uses the angles determined in Section 3.1 and rotates the qubits 0 and 1 in state $|0\rangle$ to the desired position reflecting the kinematics of the event:

```
angles=[lepPt_to_theta[event],
        lepChg_to_phi[event],
        Met_to_theta[event],
        lepEta_to_phi[event]]

def real(angles, **kwargs):
    qml.RY(angles[0], wires=0)
    qml.RZ(angles[1], wires=0)
    qml.RY(angles[2], wires=1)
    qml.RZ(angles[3], wires=1)
```

3.3 Discriminator

A quantum discriminator is designed in the same manner as an artificial Neural Network: a first layer to encode the data, a number of hidden layers with a trainable parameter and an output layer for the prediction that is going to be used in the cost function we want to minimize. Figure 13 illustrates the design of the quantum circuit discriminator.

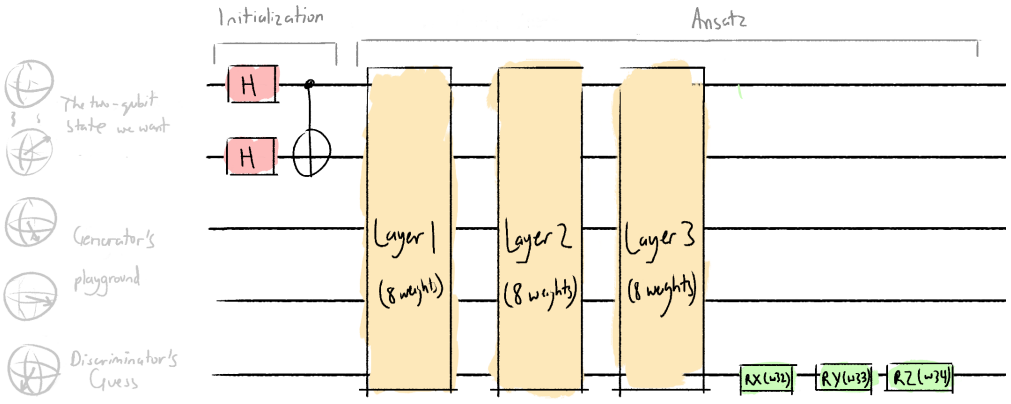

Figure 13: Design of the quantum discriminator.

The *input_layer* ensures that both qubit 0 and 1 are in a superposition and entangled. The *hidden_layer* receive a weight

(an angle) that manipulates rotations to the qubits in order to find the best set of angles that can properly distinguish between real and synthetic data. The *output_layer* rotates qubit 4 in all 3 axis before the output which will be the prediction of the discriminator. Finally, the *discriminator* circuit is defined as:

```
def hidden_layer(w, **kwargs):
    qml.RX(w[0], wires=0)
    qml.RX(w[1], wires=1)
    qml.RX(w[2], wires=4)
    qml.RZ(w[3], wires=0)
    qml.RZ(w[4], wires=1)
    qml.RZ(w[5], wires=4)
    qml.MultiRZ(w[6], wires=[0, 1])
    qml.MultiRZ(w[7], wires=[1, 4])

def discriminator(w, **kwargs):
    # input_layer
    qml.Hadamard(wires=0)
    qml.Hadamard(wires=1)
    qml.CNOT(wires=[0,1])
    # 3 hidden_layers
    hidden_layer(w[:8])
    hidden_layer(w[8:16])
    hidden_layer(w[16:24])
    # output_layer
    qml.RX(w[24], wires=4)
    qml.RY(w[25], wires=4)
    qml.RZ(w[26], wires=4)
```

3.4 Generator

The generator is a qNN that does not receive input data but outputs a 2 qubit state in qubits 0 and 1. It also has hidden layers that depend on weights. These weights will be optimized in order to "trick" the discriminator to think that the data generated is real data. Figure 14 illustrates the design of the quantum circuit generator.

The *first_layer* of the generator ensures that both qubit 0

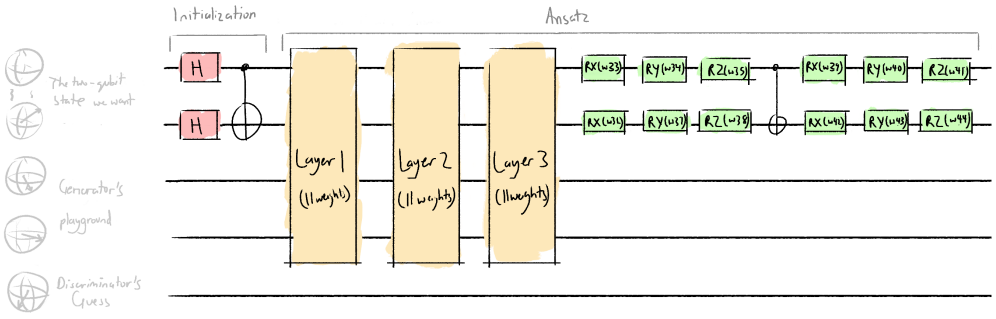

Figure 14: Design of the quantum generator.

and 1 in a superposition and entangled. The architecture for the *generator_layer* is based on [8]. The *output_layer* rotates qubit 0 and 1 in all 3 axis, entangles them, and rotates them once again. This final state of qubit 0 and 1 corresponds to the synthetic that will be fed to the discriminator. Finally, the *generator* circuit is defined as:

```
def generator_layer(w):
    qml.RY(w[0], wires=0)
    qml.RY(w[1], wires=1)
    qml.RY(w[2], wires=2)
    qml.RY(w[3], wires=3)
    qml.RZ(w[4], wires=0)
    qml.RZ(w[5], wires=1)
    qml.RZ(w[6], wires=2)
    qml.RZ(w[7], wires=3)
    qml.MultiRZ(w[8], wires=[0, 1])
    qml.MultiRZ(w[9], wires=[2, 3])
    qml.MultiRZ(w[10], wires=[1, 2])

def generator(w, **kwargs):
    # first_layer
    qml.Hadamard(wires=0)
    qml.Hadamard(wires=1)
    qml.CNOT(wires=[0,1])
    qml.Barrier(wires=[0,4],only_visual=True)
    # 3 generator lawyers
    generator_layer(w[:11])
    generator_layer(w[11:22])
```

```
    generator_layer(w[22:33])
    # output_layer
    qml.RX(w[33], wires=0)
    qml.RY(w[34], wires=0)
    qml.RZ(w[35], wires=0)
    qml.RX(w[36], wires=1)
    qml.RY(w[37], wires=1)
    qml.RZ(w[38], wires=1)
    qml.CNOT(wires=[0, 1])
    qml.RX(w[39], wires=0)
    qml.RY(w[40], wires=0)
    qml.RZ(w[41], wires=0)
    qml.RX(w[42], wires=1)
    qml.RY(w[43], wires=1)
    qml.RZ(w[44], wires=1)
```

3.5 Quantum nodes

A *QNode* contains a quantum function and the computational device it is executed on. We need to create two quantum nodes. One to train the discriminator, we feed the real data to the discriminator so that it can learn it. A second one where we feed generated data to train the generator.

```
@qml.qnode(dev, interface='tf')
def real_disc_circuit(angles, disc_weights):
    real(angles)
    discriminator(disc_weights)
    return qml.expval(qml.PauliZ(4))

@qml.qnode(dev, interface='tf')
def gen_disc_circuit(gen_weights, disc_weights):
    generator(gen_weights)
    discriminator(disc_weights)
    return qml.expval(qml.PauliZ(4))
```

Figure 15 illustrates the real+discriminator quantum circuit.

Figure 16 illustrates the generator+discriminator quantum circuit.

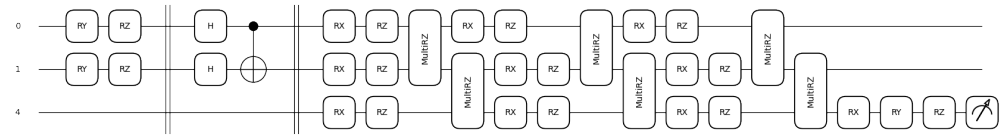

Figure 15: Real+discriminator quantum circuit.

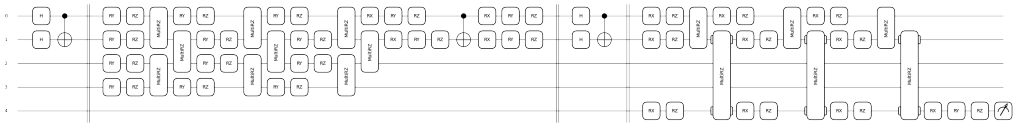

Figure 16: Generator quantum circuit.

3.6 Cost Functions

There are two cost functions of interest, corresponding to the two stages of the qGAN training. These cost functions are built in two parts: the first part is the probability that the discriminator correctly classifies real data as real (true). The second piece is the probability that the discriminator classifies generated data as real.

The discriminator is trained to maximize the probability of correctly classifying real data, while minimizing the probability of mistakenly classifying gen data:

$$Cost_D = Prob(true|\text{gen}) - Prob(true|\text{real}) \tag{25}$$

The generator is trained to maximize the probability that the discriminator accepts generated data as real:

$$Cost_G = -Prob(true|\text{gen}) \tag{26}$$

In code format:

```
def prob_real_true(disc_weights):
```

```
    true_disc_output = real_disc_circuit(phi, theta, omega,
                                         disc_weights)
    # convert to probability
    prob_real_true = (true_disc_output + 1) / 2
    return prob_real_true

def prob_gen_true(gen_weights, disc_weights):
    gen_disc_output = gen_disc_circuit(gen_weights,
                                         disc_weights)
    # convert to probability
    prob_gen_true = (gen_disc_output + 1) / 2
    return prob_gen_true

def disc_cost(disc_weights):
    cost = prob_gen_true(gen_weights, disc_weights) -
        prob_real_true(disc_weights)
    return cost

def gen_cost(gen_weights):
    return -prob_gen_true(gen_weights, disc_weights)
```

3.7 Training: Discriminator vs Generator

Both the initial weights for the discriminator and the generator are randomly initialized. At this level, the probability for the discriminator to classify real data as true is ≈90% and the probability to classify generated data as true is ≈60%. The Stochastic Gradient Descent Optimizer [15] is used to minimize both cost functions and find the best discriminator and generator weights. Next, the training functions are defined:

```
opt = tf.keras.optimizers.SGD(0.4)

def train_disc(it,disc_loss):
    cost = lambda: disc_cost(disc_weights)
    for step in range(it+1):
        opt.minimize(cost, disc_weights)

def train_gen(it,gen_loss):
    cost = lambda: gen_cost(gen_weights)
    for step in range(it+1):
```

```
        opt.minimize(cost, gen_weights)
```

The complete train of the qGAN consists in repeating 3 times a training cycle. A training cycle is defined by training the discriminator for 20 iterations and then by training the generator for 5 iterations. At the end of the complete train, the probability for the discriminator to classify real data as true is ≈83% and the probability to classify generated data as true is ≈99%. A complete train takes ≈20 minutes in my personal computer, this for a single event.

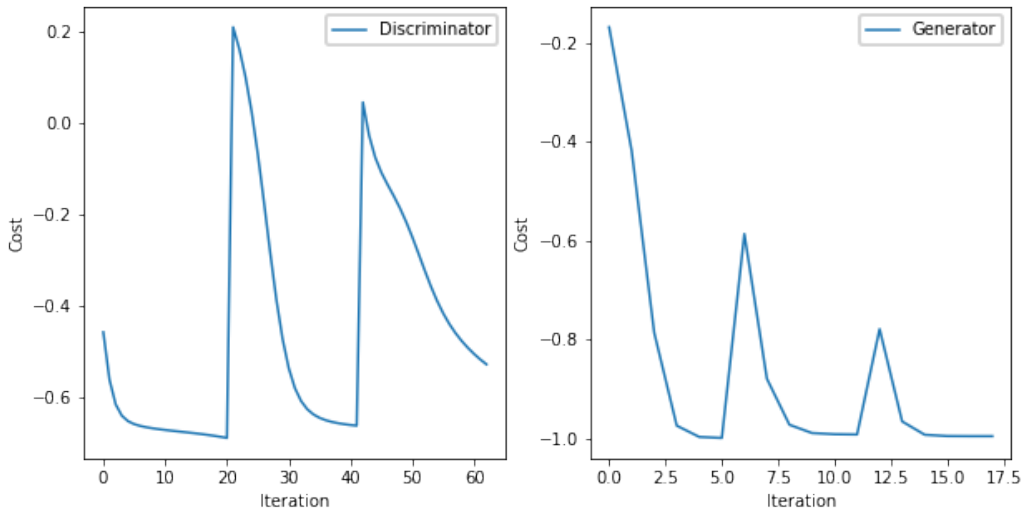

Figure 17: The cost for both the discriminator and generator as a function of number of iterations.

3.8 Results

We can now compare the real data with the generated one in Figure 18 and transform the generated qubits into the generated events kinematics. The latter is reported in Table 1.

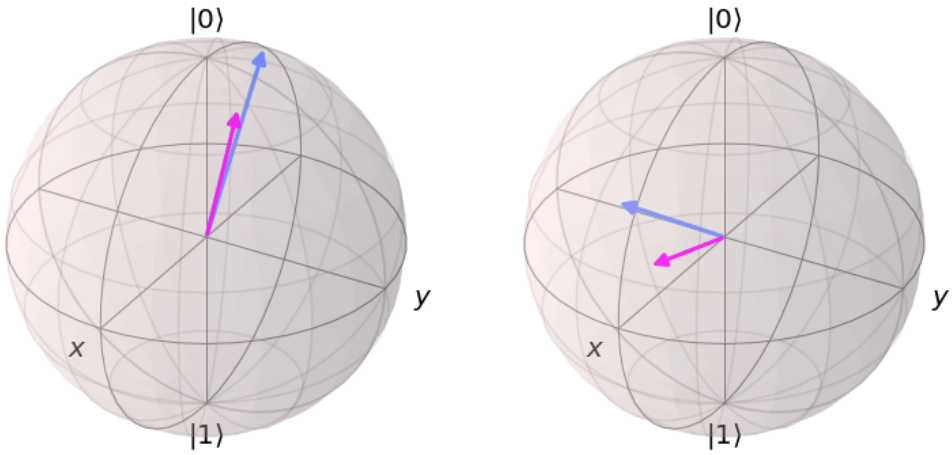

Figure 18: Comparison in the Bloch sphere of the real data (in blue) with the generated one (in purple) for both qubits 0 and 1.

Table 1: Comparison of the event kinematics between the real and generated events.

Data	$p_{\mathrm{T}}(\ell)$ [GeV]	Charge(l)	η(l)	$p_{\mathrm{T}}^{\mathrm{miss}}$ [GeV]
real	8.7	1	0.90	685
generated	13.3	1	0.95	748

3.9 tmp

4 Summary

asd

References

[1] Aad, Georges and others. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Phys. Lett. B* 716 (2012), p. 1. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].

[2] C. Balázs, M. Carena, and C. E. M. Wagner. "Dark matter, light stops and electroweak baryogenesis". In: *Phys. Rev. D* 70 (2004), p. 015007. DOI: 10.1103/PhysRevD.70.015007. arXiv: hep-ph/0403224 [hep-ph].

[3] Riccardo Barbieri, S. Ferrara, and Carlos A. Savoy. "Gauge models with spontaneously broken local supersymmetry". In: *Phys. Lett. B* 119 (1982), p. 343. DOI: 10.1016/0370-2693(82)90685-2.

[4] F. Bloch. "Nuclear Induction". In: *Phys. Rev.* 70 (7-8 Oct. 1946), pp. 460–474. DOI: 10.1103/PhysRev.70.460. URL: https://link.aps.org/doi/10.1103/PhysRev.70.460.

[5] Chatrchyan, Serguei and others. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Phys. Lett. B* 716 (2012), p. 30. DOI: 10.1016/j.physletb.2012.08.021. arXiv: 1207.7235 [hep-ex].

[6] Serguei Chatrchyan et al. "Observation of a new boson with mass near 125 GeV in pp collisions at $\sqrt{s} = 7$ and 8 TeV". In: *JHEP* 06 (2013), p. 081. DOI: 10.1007/JHEP06(2013)081. arXiv: 1303.4571 [hep-ex].

[7] *Cirq*. 2022. URL: https://quantumai.google/cirq.

[8] Pierre-Luc Dallaire-Demers and Nathan Killoran. "Quantum generative adversarial networks". In: *Phys. Rev. A* 98 (1 July 2018), p. 012324. DOI: 10.1103/PhysRevA.98.012324. URL: https://link.aps.org/doi/10.1103/PhysRevA.98.012324.

[9] Sally Dawson, E. Eichten, and C. Quigg. "Search for supersymmetric particles in hadron-hadron collisions". In: *Phys. Rev. D* 31 (1985), p. 1581. DOI: 10.1103/PhysRevD.31.1581.

[10] Glennys R. Farrar and Pierre Fayet. "Phenomenology of the Production, Decay, and Detection of New Hadronic States Associated with Supersymmetry". In: *Phys. Lett. B* 76 (1978), p. 575. DOI: 10.1016/0370-2693(78)90858-4.

[11] H. E. Haber and G. L. Kane. "The search for supersymmetry: Probing physics beyond the standard model". In: *Phys. Reports* 117 (1985), p. 75. DOI: 10.1016/0370-1573(85)90051-1.

[12] Stephen P. Martin. "A supersymmetry primer". In: *Adv. Ser. Direct. High Energy Phys.* 18 (1998), p. 1. DOI: 10.1142/9789812839657_0001. arXiv: hep-ph/9709356 [hep-ph].

[13] Hans Peter Nilles. "Supersymmetry, supergravity and particle physics". In: *Phys. Rept.* 110 (1984), p. 1. DOI: 10.1016/0370-1573(84)90008-5.

[14] *Pennylane*. 2022. URL: https://pennylane.ai/.

[15] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[16] *Searching for top squarks with CMS data*. 2022. URL: https://cms.cern/news/searching-top-squarks-cms-data.

[17] FredL. VanRossum GuidoandDrake. *Python3 Reference Manual*. ScottsValley,CA: CreateSpace, 2009. ISBN: 1441412697.

[18] J. Wess and B. Zumino. "Supergauge transformations in four dimensions". In: *Nucl. Phys. B* 70 (1974), p. 39. DOI: 10.1016/0550-3213(74)90355-1.

[19] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. "Quantum Generative Adversarial Networks for learning and loading random distributions". In: *npj Quantum Information* 5.1 (Nov. 2019). DOI: 10.1038/s41534-019-0223-2. URL: https://doi.org/10.1038%2Fs41534-019-0223-2.