UNIVERSIDAD
NEBRIJA

Escuela Politécnica Superior
Escuela Superior de Informática

Redes Adversarias Generativas Cuánticas

Máster en Computación Cuántica

Autor: **Juan José Herrera Aranda**
Director: Roberto Campos Ortiz

Madrid 2023

Redes Adversarias Generativas Cuánticas

Juan José Herrera Aranda

Juan José Herrera Aranda *Redes Adversarias Generativas Cuánticas*.
Trabajo de fin de Máster. Curso académico 2022-2023.

Responsable de tutorización Roberto Campos Ortiz

Máster en Computación Cuántica

Escuela Politécnica Superior

Universidad Antonio de Nebrija

Declaración de originalidad

D./Dña. Juan José Herrera Aranda

Declaro explícitamente que el trabajo presentado como Trabajo de Fin Máster (TFM), correspondiente al curso académico 2022-2023, es original, entendido esto, en el sentido de que no se ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Madrid a 18 de julio de 2026

Fdo: Juan José Herrera Aranda

Dedicatoria
A mi familia, en especial a mi hermano Alejandro.

Índice general

Agradecimientos

Este trabajo fin de máster es el fruto del aprendizaje de un año entero inmerso en el estudio de la computación cuántica. Agradecer primeramente a mi tutor Roberto Campos la dedicación y el tiempo que me ha dedicado, en especial por esa larga primera sesión que tuvimos online en la que me estuvo explicando el funcionamiento de las QGANs y por las correcciones milimétricas realizadas en el trabajo. Agradecer también a mis compañeros del máster por su trato cálido y amigable durante todo este año, que aunque haya sido en modalidad online en su mayor parte, estoy seguro de que no perderemos el contacto. Agradecer por supuesto a los profesores que se han implicado y esforzado en su labor docente, en especial a Álvaro Díaz y a José Javier Paulet. Y por supuesto, una especial mención en agradecimiento a mis familiares y amigos que han estado conmigo todo este año y me han dado ánimos en los momentos más duros para poder seguir adelante, gracias a todos ellos.

Resumen

Este trabajo aborda las redes adversarias generativas cuánticas (QGANs) desde una perspectiva teórica y aplicándolas a un caso práctico. Primeramente, en el capítulo 1, se abordarán las redes adversarias generativas clásicas (GANs), que son los algoritmos en los que las QGANs están inspiradas. Se explicarán teóricamente definiendo todas sus partes así como el proceso de entrenamiento y la función de coste que optimizan.

Tras comprender cómo operan las GANs y sus principales rasgos, se verá en el capítulo 2, los circuitos variacionales, que son los elementos que conforman gran parte de la anatomía de las QGANs. A lo cual le sigue el capítulo 3, dedicado principalmente a abordar uno de los aspectos críticos en el aprendizaje automático cuántico, que es la codificación de datos clásicos en sistemas cuánticos. Finalmente, en el capítulo 4, se introducirán las redes adversarias generativas cuánticas, explicando tanto sus distintas componentes como la función de coste desde un punto de vista de la mecánica cuántica.

Por último, y para poner en práctica estos algoritmos, en el capítulo 5 se detallará un estudio práctico en el que se aplican varias arquitecturas de QGANs para aprender distintas distribuciones de probabilidad bien conocidas. El objetivo es demostrar empíricamente con casos sencillos que las QGANs son capaces de operar satisfactoriamente. Estos algoritmos aún están en una fase temprana de desarrollo y la creación de QGANs que operen al mismo nivel que las GANs clásicas es aún algo inalcanzable, quedando estudios sobre problemas más complejos como algo realizable de cara a trabajos futuros.

Abstract

Quantum computing is an emerging new computing paradigm that applies the principles of quantum mechanics. Since its beginnings at the end of the last century, it has been increasingly developed to the present day. One of the most promising branches of this new paradigm is quantum machine learning. This emerging branch tries to develop artificial intelligence algorithms to solve tasks. Many of these algorithms are inspired by classically existing algorithms such as support vector machines, the KNN algorithm, neural networks or what the paper is about, generative adversarial networks (GANs). This paper deals with the quantum version of the latter algorithm. But before going into detail, and for pedagogical reasons, the reader will first be introduced to classical generative adversarial networks in chapter 1, since in order to understand the quantum version it is convenient to understand the classical one first. This chapter will give a brief introduction to diffusion models and go into detail on GANs. We will then go on to describe the different parts that make up a GAN and finally we will detail one of the key aspects of GANs, the cost function to be optimised, and then go on to explain the training process.

After understanding how GANs work and their main characteristics, some prior knowledge of variational circuits will be necessary, since QGANs are modelled on the basis of two circuits. Therefore, in the chapter 2 this knowledge will be introduced to the reader, an introduction will be given explaining what these circuits are, the different phases of a variational circuit will be explained together with the different possibilities when designing the architectures. Finally, the main problems presented by these circuits will be explained, both hardware limitations and software problems.

After understanding the latter, only one important step remains to be tackled when building these variational circuits. That is the coding of classical data in quantum systems, covered in the chapter on coding. This part is something that does not exist within classical computing, as the very nature of data does not require it. Instead, the information units of quantum computers are qubits, which have a completely different nature and properties than

bits. It is therefore necessary to adapt quantum information to these systems. The encoding of classical data in quantum systems is a critical point in many algorithms and is also a limitation at hardware level, since a quantum computer does not have as many qubits as a classical computer has bits, so the information load is very limited and has to be as efficient as possible. On the other hand, and taking into account the noise of NISQ computers, it is also required that the information loaded is robust to noise. In this chapter the main existing coding techniques will be discussed.

Finally, with all of the above, the chapter 4 introduces QGANs. This chapter will give a brief introduction to these algorithms and then proceed to explain in detail the different parts that make up QGANs. We will also discuss the possibility of hybridising classical and quantum components, for example, a QGAN in which the generator is classical and the discriminator is quantum. The optimisation problem formulated in terms of quantum mechanics will also be discussed.

To put these algorithms into practice, a practical study will be detailed in chapter 5 in which several QGANs architectures are applied to learn different well-known probability distributions. The aim is to demonstrate empirically that the generator of a purely quantum QGAN is able to learn the intrinsic characteristics of samples following a certain probability distribution and be able to generate synthetic data very similar to these samples. This study has not been taken to more realistic applications because it is unfeasible, QGANs are still at an early stage of development and the creation of QGANs that operate at the same level as classical GANs is still unachievable, remaining as something achievable for future work.

Prefacio

Los computadores clásicos de hoy en día, no dejan de ser Máquinas de Turing que operan secuencialmente. Incluso dentro del paradigma de computación clásica paralela, lo que se tiene son varias máquinas cooperando para resolver un problema pero trabajando cada una de manera secuencia. Este principio de funcionamiento, no es puramente paralelo y tiene serias limitaciones ya que para ciertas tareas, el tiempo necesario para su cálculo puede llegar a ser mayor que los años de vida que tiene el universo o incluso mayor que el número de partículas existentes en el mismo. Desde el paradigma clásico, resulta muy complicado enfrentarse a problemas que no estén en la clase de complejidad P (aquellos que pueden ser resueltos por una máquina de Turing en tiempo polinomial). En algunos tipos de problemas, como los de optimización, uno se conforma con una solución aproximada, aunque no sea exacta. Los algoritmos que atacan a estos problemas suelen explorar un pequeño espacio de soluciones mediante alguna heurística que los va guiando de algún modo u otro y que paran según un criterio de parada. No tienen manera de saber si la solución encontrada es o no la óptima.

Los ordenadores clásicos funcionan mediante una tecnología de transistores CMOSFET y muchos de los avances en la computación se debe a la miniaturización de éstas componentes que incorporan los chips. En 1965 Gordon E. Moore notó que el número de transistores en los microprocesadores se estaba duplicando cada dos años, y empíricamente formulo la conocida Ley de Moore. Sin embargo, esta tendencia no continuará indefinidamente, ya que no se podrá realizar miniaturizaciones por debajo de la escala atómica, además de otros retos complejos, como el de la disipación del calor del chip sin que se dañe debido al sobrecalentamiento que ocasiona un aumento de la densidad de transistores en un mismo volumen.

Entorno a aquellos años, la era de la computación clásica estaba en pleno desarrollo y aún quedaba mucho por hacer, aunque algunos científicos ya vaticinaban su final. Esto motivó a muchos otros investigadores a indagar en nuevos paradigmas de computación. Fue en 1982 cuando Richard Feynman, premio Nobel de Física en 1965, consideró usar sistemas cuánticos de cara

a simular sistemas en mecánica cuántica. Todo se quedo en un mero plano hipotético hasta que, unos años después, David Deutchs describió la máquina de computación cuántica universal. Finalmente, no fue hasta 1994 cuando Peter Shor demostró mediante un algoritmo que aplicaba este nuevo paradigma, que se podría resolver el problema de la factorización de números primos en tiempo polinomial, marcando el inicio de una nueva era. Poco después, en 1996 surgió uno de los algoritmo más icónico de la computación cuántica, el algoritmo de Grover, el cual fue el primer algoritmo de búsqueda en usar el principio de superposición para realizar una búsqueda totalmente paralela en un espacio de estados, algo imposible clásicamente.

Paralelamente a la época en la que Feynman propuso la idea de la computación cuántica, la inteligencia artificial venía de pasar su primer gran invierno (1974-1980), debido entre otros factores, a la desilusión popular y pérdida de inversores por no cumplir con las expectativas impuestas. Tras ese primer invierno, hubo un período álgido en donde destaca el desarrollo de los primeros sistemas expertos en 1981 y a la invención del algoritmo *Back-Propagation* en 1986, que ayudaba a calcular de manera eficiente los pesos de una red neuronal. Pero poco después, en 1987, se sumergió en otro largo y profundo invierno, el cual no finalizó hasta finales de la década de 1990, poco antes de que Shor propusiera su algoritmo de factorización. Tras la invención de varios tipos de redes neuronales como las recurrentes a finales de 1980, y otras como las LSMTs en 1997 y las convolucionales en 1998, la inteligencia artificial despertó de nuevo el interés de las personas y resurgió con más fuerza que nunca. A principios del milenio, muchas grandes empresas y los gobiernos de las principales potencias mundiales invirtieron importantes sumas de dinero en la inteligencia artificial. Poco después, en 2006 surgió el término aprendizaje automático y aprendizaje profundo. La inteligencia artificial se fue desarrollando a pasos de gigante y, poco a poco, empezaron a emerger empresas puramente dedicadas a investigarlas y a ponerlas en práctica. Destacar la invención en 2014 de las redes adversarias generativas a manos de GoddFellow. A día de hoy, la inteligencia artificial no ha parado de desarrollarse. Tal ha sido la rapidez en el progreso que hasta los gobiernos están aplicando normas que regulen su uso así como otras normas que exigen el cumplimiento de ciertos estándares éticos a la hora de la creación y el desarrollo de IAs. El futuro de la IA es cada vez más ambicioso, yendo las líneas de investigación hacia el desarrollo de inteligencias artificiales de propósito general.

De manera paralela, el estudio de la computación cuántica se centraba

principalmente en teoría de la información cuántica, criptografía cuántica y resolución de problemas de optimización. No fue hasta que definitivamente se acuñaron los términos de aprendizaje automático y aprendizaje profundo, que no surgió lo que hoy se conoce como Quantum Machine Learning. Esta rama emergente de la computación cuántica trata de desarrollar algoritmo de inteligencia artificial, pero desde el paradigma de la computación cuántica. Alguno están inspirados en algoritmos ya existentes clásicamente, como las máquinas de soporte vectorial, el algoritmo de los vecinos más cercanos, la redes neuronales y las redes generativas adversarias.

Actualmente se han podido desarrollar ordenadores que funcionan según los principios de la mecánica cuántica pero aún están lejos lograr una ventaja sobre la computación clásica, dado que tienen pocos qubits, la interconexión entre ellos es pobre y los tiempos de coherencia son muy pequeños, entre otros factores. Fundamentalmente se debe a la limitación tecnológica, la cual no permite aún la fabricación de ordenadores puramente cuánticos libre de errores en los que haya un gran número de qubits interconectados todos con todos.

1 Redes Adversarias Generativas Clásicas

Los modelos generativos son una familia de algoritmos dentro de la inteligencia artificial cuyo propósito es poder generar a partir de ruido datos sintéticos muy parecidos a datos reales que toman como referencia. El estudio y desarrollo de estos modelos ha sido un tema candente entre los investigadores desde 2014. En este capítulo se estudiará una muy conocida variedad de modelo generativo, las redes adversarias generativas, *GANs*. Este capítulo se estructura como sigue: en la sección 1.1 se dará una introducción a los modelos generativos y a las distintas variantes, haciendo hincapié en las GAN; en la sección 1.2 se estudiarán los componentes de una GAN en la sección 1.3 se analizará la función de coste usada en el entrenamiento; finalmente la sección 1.4 tratará sobre el proceso de entrenamietno de una GAN.

1.1. Modelos Generativos: GANs

Los modelos generativos son algoritmos de aprendizaje automático que intentan aproximar distribuciones de probabilidad complejas y de alta dimensionalidad a partir de muestras con el objetivo de producir nuevas muestras artificiales con las características intrínsecas a las originales, en otras palabras, intentan comprender y replicar patrones en los datos de entrada para posteriormente reproducirlo en la creación de nuevas muestras. Inicialmente fueron impulsados en el año 2006 por la aparición de las conocidas *Restricted Boltmanz Machines* (RBM) [SMH07], *Deep Belief Networks* (DBM) [HOT06] entre otras, y posteriormente estas técnicas evolucionaron y surgieron otras como *Deep Boltmanz Machines* (DBM) [Hin07] y *Deep Autoencoders* [Bal12]. El auge no fue hasta 2014 con la aparición de las Redes Generativas Adversarias (GANs) [GPAM+14] por manos de Goodfelow que empezo toda una carrera por el estudio de estos modelos. Cabe destacar que en ese mismo año nacieron los Autoencoder Variacionales [Doe16]. Desde ahí, diversos modelos fueron apareciendo y mejorando a los ya existentes como por ejemplo, *Adversarial Autoencoders* (AEEs) [MSJ+15], *BigGAN* [AB18] o las *StyleGAN* [TK19].

Los modelos generativos son usados en una amplia variedad de aplica-

ciones, desde la más conocida, como la generación de imágenes, hasta otras, como la producción de música. Hay varios tipos de modelos generativos incluyendo las combinaciones que hay entre éstos. Los más comunes son:

1. GANs (Redes Adversarias Generativas): Unas de las arquitecturas más populares. Usadas comúnmente en la generación de imágenes y en el modelado del lenguaje natural [GPAM+14].

2. VAEs (Autoencoders Variacionales): Otro tipo de modelos basados en redes neuronales. Útiles en la generación de imágenes y en la representación de datos complejos [Doe16].

3. Modelos de Difusión: Son una clase de modelos de variables latentes. Son cadenas de Markov entrenadas mediante inferencia variacional. El objetivo es aprender la estructura latente de un conjunto de datos modelando la forma en que los puntos de datos se difunden a través del espacio latente [HJA20, SDWMG15].

4. Modelos de Autoregresión: Otro tipo de técnica basada en la generación de una secuencia de valores a través de una distribución de probabilidad condicionada. Muy útiles en la generación de texto y música. [UML13, VDOKK16]

5. Redes Generativas de Flujo: Arquitectura reciente basada en la transformación de un espacio latente a un espacio de datos usando una red neuronal. Muy efectivas en la generación de imágenes de alta calidad [RM15, TT13].

En lo que sigue, se abordará un tipo muy concreto de modelo generativo, las redes adversarias generativas (GANs) [GPAM+14]. Su aparición marcó un punto de inflexión ya que la tarea de aproximar distribuciones de probabilidad intratables resultaba realmente afanoso [Pea63], produciendo cierta reticencia a la hora de lanzarse en esta área de investigación habiendo otras más prósperas en aquel momento como por ejemplo, la clasificación o la visión por computador. A partir de su aparición, hubo un avance casi exponencial en esta área.

Una Red Adversaria Generativa (GAN) consiste en dos modelos de redes neuronales llamados discriminador y generador, denotados respectivamente con las letras D y G. Antes de entrar en detalle con la explicación y fuera de todo rigor, se dará un sencillo ejemplo a modo pedagógico de su funcionamiento. Suponga que usted es un gran pintor y un estafador, su objetivo es

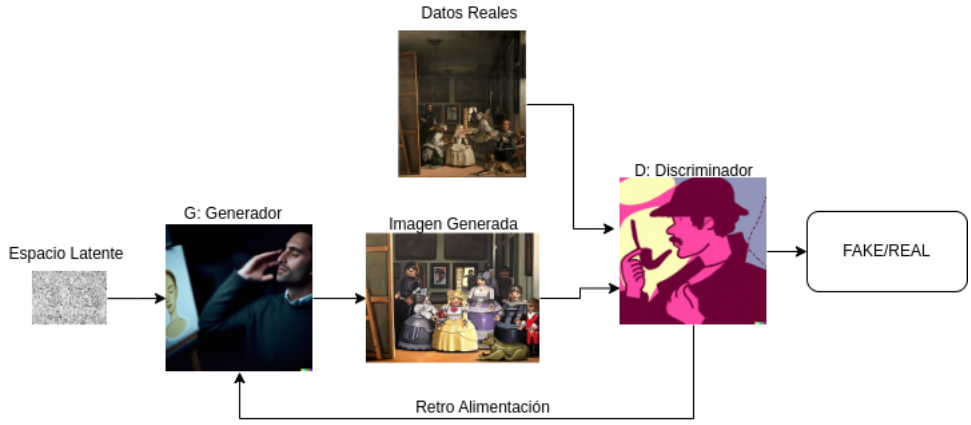


Figura 1.1: Esquema visual del proceso de generación de cuadros de Velazquez.

vender cuadros de Diego Velázquez falsos a un precio desorbitado abogando que son realmente originales. Para ello busca un cómplice especializado en los cuadros del pintor andaluz, el cual puede distinguir con facilidad qué obras son las originales y cuáles son una réplica. Usted tendrá el papel del generador, es decir, pintará obras haciendo que parezcan que son del propio Velázquez. Para ello, recibirá una retroalimentación de su cómplice, el discriminador, que juzgará si realmente son o no del propio pintor y le irá guiando en el proceso. Tendrá que conseguir que dicho experto no sea capaz de distinguir si el cuadro realmente está pintado por usted o por Velázquez. El propósito no es que usted genere una obra que sea una réplica de la original, sino que adquiera el propio estilo innato de Diego Velázquez - adelantando un poco la materia, el hecho de que usted adquiera el propio estilo innato del pintor hace referencia a que la distribución de probabilidad del generador reproduzca o se aproxime todo lo que pueda a la distribución de probabilidad original - y sea capaz de pintar obras que realmente parezcan pintadas por él. Una vez adquiera esa habilidad, será capaz de producir obras que contengan las características intrínsecas del modo de pintar de Velázquez y pueda engañar a cualquier otro experto del tema. De este modo, cumplirá su objetivo y venderá los cuadros, que además son únicos por estar pintados por usted, a un precio desorbitado, generando mucha polémica al respecto sobre cómo esas obran han pasado desapercibidas tras tres siglos después del fallecimiento del pintor.

Las GANs se caracteriza por entrenar simultáneamente dos modelos de conocimiento sobre un conjunto de datos específico: un modelo generativo

G, que genera nuevas muestras - datos falsos - a partir de un espacio latente con ruido y las cuales tendrán una distribución subyacente que se aproxime tanto como pueda a la original; y un modelo discriminador D, que estima la probabilidad que tiene una muestra de ser real en vez de falsa. El objetivo del generador es minimizar la probabilidad de clasificar los datos falsos como reales, mientras que el de G es maximizar la probabilidad de que D se equivoque al clasificar las muestra que él genera. Cabe a destacar que cuanto mejor sea el discriminador, mejor será la calidad del generador. En una GAN el caso ideal no sería que el generador produzca datos con una distribución de probabilidad idéntica a la de los datos de entrenamiento, ya que en ese caso se estaría copiando exactamente la distribución de datos del conjunto de entrenamiento, pero dicha distribución no tiene por qué ser la objetivo. Es decir, dentro de todo el espacio de soluciones, los datos de entrenamiento es un subconjunto que pretende ser lo suficientemente general como para ser una muestra representativa del espacio en su globalidad, pero en la práctica esto no es así del todo. Por lo que reproducir exactamente la distribución del conjunto del entrenamiento no implicaría alcanzar realmente la distribución objetivo. Una situación ideal para una GAN sería ser capaz de capturar y entender las características intrínsecas y principales de la distribución del conjunto de entrenamiento para poder generar datos con esas mismas características provocando que el discriminador sea incapaz de detectar si la muestra es real o generada y obtenga un 50 % de precisión en sus predicciones. En este punto se acaba el entrenamiento y se dice que se ha llegado a un *equilibrio NASH*, nombre proveniente de la teoría de juegos.

1.2. Anatomía de las GANs

En esta sección se va a profundizar en la explicación de las distintas partes que componen una GANs. De aquí en adelante y por simplicidad se supondrá que las muestras son vectores de una componente espacial, $x \in \mathbb{R}^N$. Si se quisiera generalizar para el caso de imágenes se considerarían muestras $x \in \mathbb{R}^{h \times w \times c}$ donde h, w y c corresponden a la altura, anchura y canales de la imagen respectivamente.

Las componentes de una GAN se encuentran incrustadas en la imagen 1.2 y se observan cuatro grandes partes: espacio latente, datos de entrenamiento, generador y discriminador.

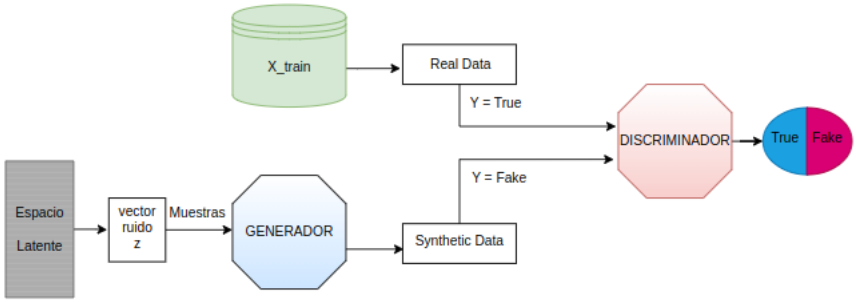


Figura 1.2: Esquema anatómico de una GAN estándar.

1.2.1. Espacio Latente

El primer elemento es el espacio latente, de aquí se van a muestrear los vectores aleatorios que le sirven de entrada al generador para producir muestras sintéticas. Es importante destacar que las distribuciones de probabilidad de los vectores no están especificadas de antemano, son arbitrarias. No es sino el generador quien aprende una distribución de probabilidad usando las muestras extraídas del espacio latente. Esta distribución de probabilidad aprendida es lo que se utiliza para mapear nuevos vectores aleatorios z en muestras sintéticas. Matemáticamente el espacio latente se puede modelar de la siguiente manera

Definición 1.1 (Espacio Latente)**.** Se define el espacio latente $\mathcal{Z}$ como el conjunto de todas las posibles funciones de densidad de probabilidad $p(z_1, z_2, \ldots, z_d)$ (las cuales caracterizan unívocamente una distribución de probabilidad p_z) que describen la probabilidad conjunta de que la variable aleatoria multivariada $Z = (Z_1, Z_2, \ldots, Z_d)$ tome valores en un espacio de d dimensiones.

Una muestra obtenida del espacio latente $\mathcal{Z}$ viene dada como

$$Z = \{z \in \mathbb{R}^d : z \sim p_z \; con \; p_z \in \mathcal{Z}\}$$

Esta definición indica que $\mathcal{Z}$ es un espacio de distribuciones de probabilidad, donde cada elemento corresponde a una distribución de probabilidad arbitraria que define la forma en la que se deben muestrear los vectores aleatorios. En general, se asume que el espacio latente de una GAN sigue una distribución de probabilidad multivariada.

Computacionalmente no se puede modelar dicho espacio latente, pues se

necesitaría aleatoriedad la cual no es un fenómeno admisible por la computación clásica. En su lugar y por simplicidad, se considera $\mathcal{Z} = \{\mathcal{N}(0_d, I_d)\}$ como una distribución normal multivariada de dimensión d (también se suele usar una distribución uniforme en $[0,1]^d$). De este modo, la muestra extraída de aquí quedaría como sigue,

$$Z = \{z \in \mathbb{R}^d : z \sim \mathcal{N}(0_d, I_d)\}.$$

1.2.2. Conjunto de entrenamiento

El segundo elemento es el conjunto de entrenamiento el cual contiene los datos reales y se puede definir como sigue.

Definición 1.2 (Conjunto de entrenamiento)**.** Sea $\mathcal{X} \subset \mathbb{R}^n$ una población que contiene a todos los posibles vectores de características generados según una distribución de probabilidad desconocida pero fija a lo largo del tiempo, p_{data}. Sea $\mathcal{Y}$ un conjunto que contiene a todas las posibles etiquetas. Se define el conjunto de entrenamiento, S, como un subconjunto de cardinal finito del producto cartesiano de $\mathcal{X}$ e $\mathcal{Y}$. Esto es, S es un conjunto de entrenamiento cuando $S \subset \mathcal{X} \times \mathcal{Y}$ y cumpliendo que $|S| < \infty$. La proyección en la primera componente se denotará como X_{train} y la proyección en la segunda componente como Y.

La distribución de probabilidad con la que se generan los datos reales, p_{data}, es aquella a la que el generador aspira aproximar. Para ello, se considera un etiquetado binario $\mathcal{Y} = \{0,1\}$ para diferenciar las muestras provenientes de p_{data} y de p_g, asignándoles la etiqueta 1 y la 0 respectivamente.

1.2.3. Generador

Definición 1.3 (Generador)**.** Sea p_{data} una distribución de probabilidad objetivo. Sean $k, d, n \in \mathbb{N}$ tal que $d < n$ representando la dimensión del vector de pesos, la dimensión del espacio latente y la dimensión de la salida del generador respectivamente. Sea $Z \subset \mathbb{R}^d$ una muestra extraída de $\mathcal{Z}$. Se define el generador como aquella red neuronal que implementa una función parametrizada diferenciable $G : Z \times \mathbb{R}^k \to \mathbb{R}^n$ donde $\theta = \{\theta_1, \ldots, \theta_k\}$ es el conjunto de parámetros de la red neuronal. La función toma vectores aleatorios con una distribución de probabilidad p_z y los mapea en elementos $x = G(z;\theta)$ que siguen una distribución de probabilidad conjunta $p_g(x,z) = p_z(z)p_g(x|z)$

con el objetivo de que tras el entrenamiento se consiga que $p_g \approx p_{data}$. Además, ha de verificar que fijando la segunda variable, la función $G(z;\cdot)$ sea inyectiva.

En esta definición , se considera la función que implementa la red generadora como una función de dos variables, en la que la primera representa la entrada que recibe del espacio latente y la segunda el vector de pesos de la red que es lo que se va a pretender optimizar. Por simplicidad, algunas veces se escribirá $G_\theta(z)$ o simplemente $G(z)$ o G en el caso de que se esté hablando del comportamiento de la red neuronal como ente generador de datos, sin prestar atención a los pesos.

Observación 1.1. Es importante notar que la función ha de ser inyectiva en la primera variable, esto es, fijados los parámetros de la red neuronal, si para todo $z_1, z_2 \in Z$ con $z_1 \neq z_2$, entonces $G(z_1;\cdot) \neq G(z_2;\cdot)$. Esto viene a decir que dadas dos entradas que sean iguales, la salida ha de ser la misma, en otras palabras, que a una entrada no se le correspondan dos salidas distintas. Conviene destacar esto ya que viene a decir que el generador es determinista, fijado los pesos en un instante de tiempo, bajo las mismas entradas por parte del espacio latente, las muestras sintéticas generadas no variarán.

En cada paso de entrenamiento de una GAN, se extrae una población o muestra Z del espacio latente $\mathcal{Z}$ que tendrá la función de minilote. Cada elemento $z \in Z$ sigue una distribución $z \sim p_z$, aunque por el momento, no se le dará tanta importancia. El generador es una red neuronal que toma como entrada un elemento z y produce una muestra $x = G(z)$ en el espacio de los datos.

La distribución de probabilidad del generador describe cómo los datos generados por el generador se relacionan con el espacio latente utilizado para generarlos. Se define como la distribución de probabilidad conjunta entre los datos generador por el generador y el espacio latente,

$$p_g(x,z) = p_z(z)p_g(x|z)$$

donde $p_g(x|z)$ es una función de densidad condicionada y representa la salida de los datos generados por G, $x = G(z)$, condicionados a un valor específico del espacio latente. La distribución p_g no se conoce explícitamente en la práctica y se estima a partir de muestras generadas $\{x_1, \ldots, x_n\}$ tomando el

promedio empírico de la distribución de probabilidad condicionada,

$$p_g \approx \frac{1}{n} \sum_i p_z(z_i) p_g(x_i|z_i)$$

donde z_i es la muestra usada para generar la muestra x_i. Esta estimación se puede usar para evaluar la calidad del generador.

1.2.4. Discriminador

Definición 1.4. Se define el discriminador de una GAN como una red neuronal que implementa una función diferenciable parametrizada $D : \mathbb{R}^n \times \mathbb{R}^m \to [0, 1]$ por un conjunto de parámetros $\theta = \{\theta_1, \ldots, \theta_m\}$ que actúa de medida de probabilidad.

El discriminador de una GAN es una red neuronal que tiene el objetivo de distinguir si su entrada es una muestra generada por el generador o es una muestra de la población real de los datos. Más concretamente, tiene que distinguir si la una muestra $x \in \mathbb{R}^n$ pasada como entrada presenta la distribución real de los datos, es decir, si se cumple que $x \sim p_{data}$. Para ello elabora una salida escalar que representa la estimación de probabilidad condicionada $P[x \sim p_{data}|x]$. Por tanto, el discriminador será una función de clasificación binaria que toma una entrada x cumpliendo que o bien $x \sim p_g$ o bien $x \sim p_{data}$ y elabora una salida que se interpreta como la probabilidad de que la distribución de la que provenga sea p_{data}. Así, si la salida es cercana a 1, se considera que la muestra es real, mientras que si es cercana a 0, significará que la muestra fue generada por el generador.

Teniendo en cuenta lo anterior, no resulta complicado inferir que durante el entrenamiento de la GAN, el discriminador tiene que maximizar la probabilidad de clasificar correctamente las muestras reales de las generadas, compitiendo contra el generador, el cual se encarga de minimizar la probabilidad de que éste acierte. El discriminador se denota por $D(x; \theta)$, aunque más naturalmente se hará como $D_\theta(x)$, D_θ o simplemente D si no se presta importancia a la entrada.

1.3. Función de coste

El objetivo de este apartado será realizar la construcción de la función de conste de una GAN arbitraria y entender el problema de optimización a tratar.

Se parte de la conocida función de entropía binaria cruzada

$$\mathcal{L}(\hat{y}, y) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \tag{1.1}$$

donde $y \in \{0, 1\}$ es la etiqueta, se tendría $y = 1$ si la muestra proviene de $p_{data}(x)$ e $y = 0$ si viene de $p_g(z)$, $\hat{y} \in \{0, 1\}$ es la salida del generador.

En el entrenamiento del discriminador se usan muestras tanto reales como generadas. Tomando una muestra real $x \in \mathcal{X}$ se tendría $y = 1$ e $\hat{y} = D(x; \theta)$, que sustituyendo en la ecuación (1.1) quedaría

$$L(D(x; \theta_d), 1) = \log(D(x; \theta_d)) \tag{1.2}$$

ahora tomando una muestra generada, $G(z; \theta_g)$ con $z \sim p_z$ donde $p_z \in Z$ quedaría,

$$L(D(G(z; \theta_g); \theta_d), 0) = \log(1 - D(G(z; \theta_g); \theta_d)) \tag{1.3}$$

En este punto, el objetivo del discriminador es clasificar las muestras reales y falsas correctamente. Por tanto, y teniendo en cuenta que $\log :]0, 1]) = (]-\infty, 0])$, se debe maximizar la función de coste dada por la suma de las ecuaciones (1.2) y (1.3), esto es,

$$L^{(\theta_D)} = \max \Big[log\big(D(x; \theta_d)\big) + log(1 - D\big(G(z; \theta_g); \theta_d)\big) \Big] \tag{1.4}$$

Maximizar (1.2) es equivalente a que el discriminador clasifique un dato real como real, mientras que maximizar (1.3) equivale a que clasifique correctamente un dato falso como falso. De este modo, la primera parte penalizaría al discriminador por clasificar incorrectamente muestras reales mientras que la segunda lo penalizaría por clasificar incorretamente muestras generadas.

Cabe notar que el objetivo del generador es competir contra el discriminador. Por tanto, su propósito es minimizar el problema de optimización dado en (1.4), mediante la siguiente ecuación (1.5)

$$L^{(\theta_G)} = \min \Big[log\big(D(x;\theta_d)\big) + log(1 - D\big(G(z;\theta_g);\theta_d)\big)\Big] \tag{1.5}$$

De este modo, quedaría el siguiente problema min-max

$$L = \min_{\theta_G} \max_{\theta_D} \Big[log\big(D(x;\theta_d)\big) + log(1 - D\big(G(z;\theta_g);\theta_d)\big)\Big] \tag{1.6}$$

Hay que notar que la ecuación (1.6) está destinada a una única muestra de entrenamiento, por tanto se toma la esperanza matemática para tener en cuenta el conjunto de entrenamiento completo.

$$\min_{\theta_G} \max_{\theta_D} V(\theta_D;\theta_G) :=$$
$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{x\sim p_{data}(x)}[\log D(x;\theta_d)] + \mathbb{E}_{x\sim p_z(z)}[log(1 - D(G(z;\theta_g);\theta_d))].$$

Para concluir esta sección, comentar que no hay una única función de coste en todo el proceso. El discriminador y el generador definen cada uno su respectiva función.

1.4. Entrenamiento

En una GANs, tanto el discriminador como el generador están en constante competencia. El objetivo final es que el generador produzca muestras que engañen al discriminador haciéndolas prácticamente indistinguibles de las reales. Se pretende que cada red se entrene en función de la retroalimentación de su adversario, de ahí la competitividad. El entrenamiento se divide en dos grandes fases.

En la primera fase, se congelan los pesos del generador y se actualizan los del discriminador. La entrada del discriminador es un conjunto de datos conteniendo tanto las muestras extraídas de $p_{data}(x)$ como de $p_g(z)$. Después se realiza un proceso de optimización típico de una red neuronal. Primero, partiendo de la entrada se aplica una propagación hacia adelante para obtener una salida. Con dicha salida se calcula el valor de la función de pérdida y mediante el *backpropagation* se realiza un cálculo del gradiente de dicha función que servirá a la hora de actualizar los pesos según la técnica de optimización utilizada.

En la segunda parte, se congelan los pesos del discriminador y se actualizan los del generador. Para ello se toma un vector aleatorio z obtenido del espacio latente con distribución de probabilidad $p_z(z)$ que sirve de entrada a la red generadora. Se aplica una propagación hacia adelante para generar la muestra sintética la cual es pasada al discriminador. Con la salida del discriminador se calcula la función de pérdida a la que de nuevo, aplicando *backpropagation* se obtendrá su gradiente para finalmente actualizar los pesos del generador.

El algoritmo de entrenamiento queda reflejado en 1. Cabe señalar que el papel clave para el éxito de la GAN lo juega el discriminador. Si no es capaz de diferenciar las muestras reales y las generadas, el generador no recibirá una retroalimentación útil y no será capaz de mejorar. Además, al principio del entrenamiento, el discriminador no estará bien parametrizado y los datos proporcionados por el generador tampoco serán muy buenos. Debido a ello, es conveniente especializar al discriminador a mayor razón que su adversario para posteriormente inducirle una retroalimentación de calidad. Cuanto más especializado esté el discriminador, de mayor calidad serán las muestras sintéticas proporcionadas por el generador. Con todo esto en mente, se observa cierta dependencia del rendimiento del generador con respecto a cómo de bien trabaje el discriminador. Por este motivo, el discriminador es actualizado a razón de $k : 1$ con $k \in \mathbb{N}$ con respecto al generador. Entrando más en detalle, los autores de esta técnica explican que esto hace que D se mantenga cerca de su solución óptima, siempre y cuando G cambie lo suficientemente despacio.

Data: OPT: Método de optimización
for *numero de iteraciones* **do**
  for *k pasos* **do**
    $MiniBatch_g \leftarrow \{z^{(1)}, ..., z^{(m)}\} \sim p_g(z)$;
    $Minibatch_{data} \leftarrow \{x^{(1)}, ..., x^{(m)}\} \sim p_{data}(x)$;
    $\theta_D \leftarrow UpdateWeights(L, \theta_D)$
  end
  $MiniBatch_g \leftarrow \{z^{(1)}, ..., z^{(m)}\} \sim p_g(z)$;
  $\theta_G \leftarrow UpdateWeights(L, \theta_G)$
end

Algorithm 1: Entrenamiento de una GAN

2 Circuitos variacionales

Los circuitos variacionales (VC), también conocidos como circuitos cuánticos parametrizados, son un algoritmo híbrido cuántico-clásico que tiene un importante papel dentro de la computación cuántica, más concretamente, en el aprendizaje automático cuántico (*Quantum machine Learning, QML)*. En este capítulo se tratarán los circuitos variacionales entendidos como modelos de QML. Pese a que aún no hay una demostración teórica que avale el potencial de estos algoritmos, se intuye que tendrán mucho potencial de cara al futuro. Hoy en día la demostración de supremacía cuántica es uno de los principales objetivos de la comunidad investigadora. Lo que se busca en la práctica es conseguir algún tipo de ventaja cuántica en algunas aplicaciones [DBK+22]. En la sección 2.1 se introducirán los circuitos variacionales. Posteriormente, en la sección 2.2 se verán las distintas fases de los VCs. La sección 2.3 tratará sobre las arquitecturas de un VC y finalmente, la sección 2.1 detallará algunas de las principales dificultades.

2.1. Introducción

Los circuitos cuánticos parametrizados o circuitos variacionales (VC) pueden ser vistos como modelos de aprendizaje automático con una gran capacidad expresiva. La tarea de aprendizaje describe el proceso iterativo de aprender los parámetros adecuados del circuito cuántico usando un algoritmo de optimización clásico que minimiza una función de coste de acuerdo a la salida del circuito variacional. De cara a la era NISQ, si se quiere probar un algoritmo, éste tiene que estar adaptado a las capacidades del hardware de los ordenadores cuánticos actuales. En este contexto los circuitos variacionales dan una manera efectiva de implementar estos algoritmos de cara a demostrar cierta supremacía cuántica dentro de las posibilidades de la era NISQ. En [LBR17, HM17] se puede ver que bajo algunos supuestos teóricos de complejidad bien aceptados, la clase de algoritmos VCs no puede simularse eficientemente en tiempo polinómico con recursos clásicos. Esto es notable en los simuladores cuánticos actuales en donde el más potente puede simular un sistema de hasta 50 qubits aproximadamente, pero no más. Por lo que

ya se puede ver el potencial que tendrían sistemas con un mayor número de qubits ejecutados en computadores cuánticos.

Estos circuitos están compuestos generalmente por un conjunto fijo de puertas, como por ejemplo, puertas CNOTs o rotaciones sobre qubits. Algunos VCs son capaces de generar soluciones no triviales bastante buenas a algunos problemas sin requerir de mucha profundidad.

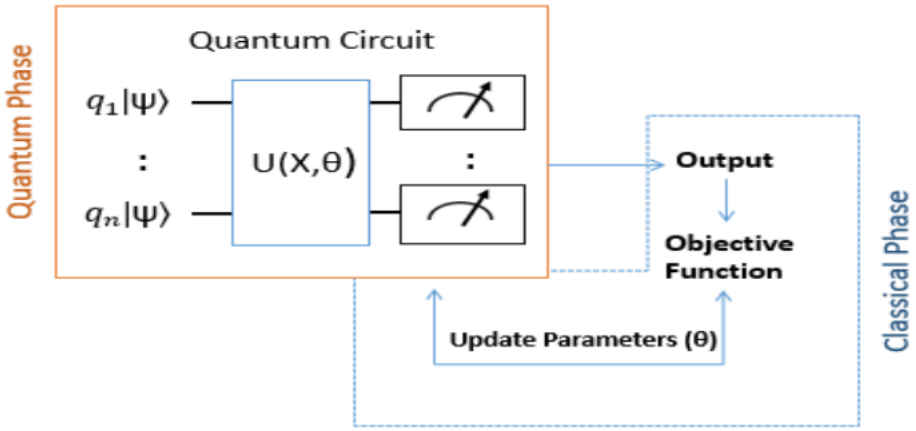


Figura 2.1: Esquema que muestra las fases cuánticas y clásicas de un circuito variacional. La parte cuántica incluye un circuito cuántico que consiste en la preparación del estado, operaciones en el circuito y la medición. La parte clásica incluye el preprocesado de la salida del circuito, la función de coste y algoritmos de aprendizaje que actualizan los parámetros de las puertas.

Los circuitos cuánticos parametrizados se incorporan dentro de un marco híbrido (Fig. 2.1), lo que resulta ideal en esta era NISQ, ya que la parte cuántica se encarga de las rutinas que son intratables por la parte clásica. En particular se encarga de preparar un estado cuántico determinado mediante la aplicación de operadores y puertas cuánticas, y de realizar el proceso de medición de dicho estado. La parte clásica se encarga de preprocesar la salida y de ejecutar un algoritmo de optimización que actualiza los parámetros del modelo. Las tareas que realiza la parte clásica suelen hacerse de manera eficiente y suponen una disminución de recursos importante en lo que respecta a la parte cuántica. De este modo, se aprovechan las capacidades de los ordenadores cuánticos actuales sin llegar a colapsarlos.

2.2. Fases en un modelo de circuito variacional

En lo que sigue se supone un sistema cuántico cerrado de n-qubits siendo $\mathbb{C}^{2^n}$ su respectivo espacio de Hilbert en el que se encuentran todos los posibles estados del sistema. Básicamente se parte del estado de la base computacional dado por el producto tensorial $|0^{\otimes n}\rangle$. A ese estado se le aplica un operador unitario produciendo un nuevo estado $U|0^{\otimes n}\rangle$. Aquí ya se puede medir el valor de un observable.

Observación 2.1. Recordar que los observables físicos están asociados con un operador $M = \sum_i \lambda_i P_i$ donde λ_i es el i-ésimo valor propio y P_i es el proyector correspondiente al subespacio propio asociado. La regla de Born establece que el resultado de una medición corresponde a uno de los autovalores y sigue la distribución de probabilidad dada por

$$p(\lambda_i) = tr(P_i U|0^{\otimes n}\rangle\langle 0^{\otimes n}|U^\dagger)$$

Introduciendo lo anterior en la definición del valor esperado se tiene que

$$\langle M\rangle = \sum_i \lambda_i p(\lambda_i) = tr(MU|0^{\otimes n}\rangle\langle 0^{\otimes n}|U^\dagger)$$

A continuación se mostrará todo el proceso de ejecución de un algoritmo variacional el cual queda bien ilustrado en la figura 2.2:

1. Se aplica un preprocesado clásico a los datos del conjunto de datos S con objeto de poder meterlos en el siguiente paso dentro del sistema cuántico. Es fundamental que los datos estén normalizados.

2. Se aplica sobre el estado inicial del sistema $|0^{\otimes n}\rangle$ un circuito de codificación, $U_{\phi(x)}$ que implementa un mapa de característiacas $\phi(x)$, también conocido como *feature map* o *encoder circuit*, para introducir los datos clásicos en el sistema cuántico. En este punto el estado del sistema viene dado por
$$|\psi_i\rangle = U_{\phi(x)}|0^{\otimes n}\rangle$$
Esta parte está más detallada en el capítulo 3.

3. Se aplica un circuito variacional, U_θ siendo θ los parámetros del circuito, con la posibilidad de actuar sobre registros de qubits adicionales. El

estado del circuito quedaría entonces

$$|\psi_f\rangle = U_\theta |\psi_i\rangle = U_\theta U_{\phi(x)} |0^{\otimes n}\rangle$$

4. Se aplica un proceso de medición en el que se estiman un conjunto de valores esperados $\{\langle M_k \rangle_{x,\theta}\}_{k=1}^{K}$. El número de repeticiones requeridos para la estimación de cada término se determina en función de la precisión que se desea tener.

5. Se aplica una función f de post-procesado clásica para procesar la salida. Por ejemplo, dentro de una tarea de regresión dicha función podría ser $f = \sum_k w_k \langle M_k \rangle_{x,\theta}$ donde w_k serían parámetros adicionales.

Se repiten los pasos (2),(3),(4) y (5) de manera iterativa durante el entrenamiento del circuito. Un algoritmo clásico de optimización se encarga de evaluar una función de coste de acuerdo a la salida del circuito y actualizar los parámetros de cara a la siguiente iteración.

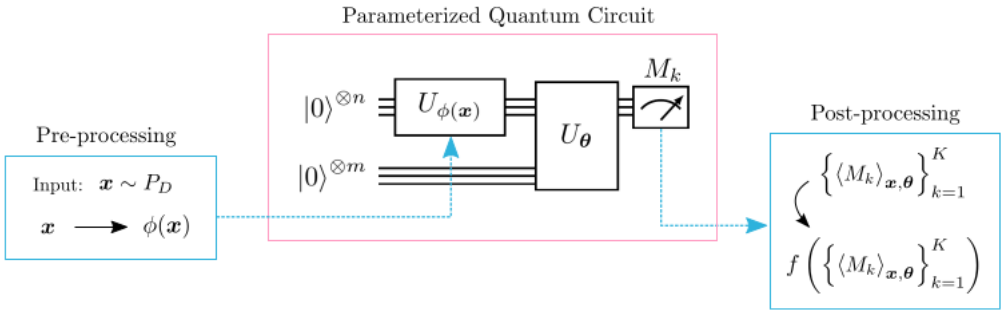


Figura 2.2: Esquema donde se aprecian las partes fundamentales de un circuitos variacional. En la parte del preprocesado se toma una muestra de datos es según una distribución de probabilidad $x \sim P_S$. Tras lo cual se codifica en un vector $\phi(x)$ que parametriza un mapa de características dentro del esquema variacional, $U_\phi(x)$. Un circuito variacional parametrizado, U_θ, con vector de pesos θ se ejecuta sobre el sistema de n qubits con los datos ya codificados y sobre un posible registro ancilla de m qubits. Finalmente se estiman una serie de observables $\{\langle M_k \rangle_{x\theta}\}_{k=1}^{K}$ mediante mediciones y se post-procesan aplicando alguna función de coste, clasificadora, regresora, etc. [BLSF19]

2.3. La arquitectura de un circuito variacional

Actualmente en la literatura científica hay gran cantidad de VCs propuestos [ZLX+22, CYQ+20, LWX+20, ZMS+23], aunque el rendimiento y la precisión de cada circuito va a depender por lo general del problema en el que se aplica. En este sentido la arquitectura del circuito es crítica. Principalmente hay tres maneras de construir arquitecturas para los VCs:

1. Arquitecturas de puertas en capa: Estas arquitecturas son muy similares a la que usan las redes neuronales clásicas. Son unas arquitecturas de capas en la que cada capa está compuesta por una serie de puertas. Las capas pueden repetirse por lo que el número de capas a usar es un hiperparámetros. El circuito se descompone en unitarios y dependiendo de la parametrización de los bloques, éstos a su vez se clasifican en subtipos (Fig. 2.3).

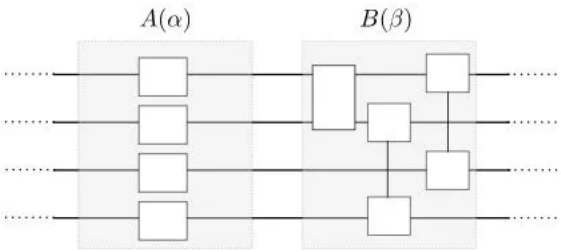


Figura 2.3: Ejemplo de la estructura de puertas en capas y la arquitectura de Ansat de operador alterno. La principal diferncia entre ambos tipos es la naturaleza de los unitarios. Fuente: *Medium*

2. Ansatz de operador alterno: Estas arquitecturas son similares a las arquitecturas de puertas en capas, ya que opera por bloques. La diferencia radica en que los unitarios que representan los bloques están definidos por Hamiltonianos que evolucionan con el tiempo. La idea es similar al paradigma de computación cuántica adiabática. (Fig. 2.3)

3. Redes Tensoriales: Estas arquitecturas no constan de capas como las anteriores, sino de una estructura fija. Se basan en la idea de descomponer un estado cuántico en términos de tensores, permitiendo así una representación compacta y un cálculo eficiente. (Fig. 2.4)

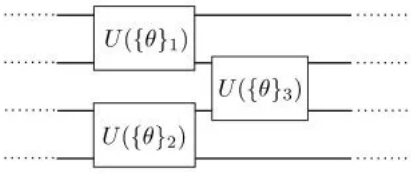


Figura 2.4: Un ejemplo de arquitectura por medio de redes tensoriales. Fuente: *Medium*

2.4. Principales dificultades

En esta sección se presentarán algunas de las principales dificultades a la hora del diseño de los VCs.

Profundidad del circuito

La primera gran dificultad es relativa a la profundidad del circuito. De manera análoga al teorema de aproximación universal de las redes neuronales [Cyb89] siempre existe un circuito cuántico que puede representar una función objetivo con un error arbitrario. El Teorema de aproximación universal expone que la salida de una red neuronal de una capa oculta, N neuronas (sin sesgo), m entradas y en donde todas las salidas de las neuronas usan como función de activación la función sigmoidal a excepción de la última, que aplica solo la identidad y que viene dada por

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(w_j^T x + \theta_j)$$

donde $w_j \in \mathbb{R}^m$ y $\alpha, \theta \in \mathbb{R}$ es densa en el espacio de las funciones continuas real valuadas en el cubo unitario n-dimensional, $C(I_n)$ (espacio Hausdorff localmente compacto). Generalizaciones de este teorema extienden el dominio a uno más amplio.

En lo relativo a la parte cuántica, el problema está en que dicho circuito puede ser exponencialmente profundo haciéndolo inviable en la práctica. Algunas investigaciones ([LTR17]) sostienen que, si el conjunto de datos está muestreado de sistemas físicos y tiene propiedades como simetrías o relaciones cuánticas, no es necesario un circuito tan costoso para obtener un resultado satisfactorio, con uno simple bastaría. Sin embargo, no se afirma lo mismo otros conjuntos de datos de otras naturalezas. Dicho esto, el circuito

variacional tiene como finalidad implementar una función que sea una aproximación de la función objetivo sin dejar de ser, por supuesto, escalable en el número de parámetros y en la profundidad.

A la hora del diseño de circuitos hay que tener en cuenta que si se quiere poner en práctica, éste ha de cumplir con el hecho de que el hardware NISQ tiene pocos qubits y suele operar en un grafo de conectividad qubit a qubit disperso y con puertas sencillas. Las puertas que se suelen usar son aquellas que produzcan algún tipo de entrelazamiento y rotaciones de un solo qubit. A partir de algunas de ellas, obviamente se puede implementar cualquier puerta mediante distintas combinaciones.

El problema de la no-linealidad

El éxito de las redes neuronales y el aprendizaje profundo ha sido la principal fuente de motivación e inspiración para el diseño de estos circuitos. De hecho, tanto los circuitos variacionales como las redes neuronales pueden entenderse como pequeñas unidades de cómputo conectadas y controladas por unos parámetros ajustables, llevando así a muchos autores a nombrarlos como redes neuronales cuánticas, *QNN*.

Por una parte, las funciones de activación en las redes neuronales son funciones no lineales y son necesaria para garantizar una buena aproximación y la universalidad, [Cyb89]. Esto se contrapone en cierto modo las operaciones en la computación cuántica que tienen que ser unitarias y lineales para preservar la probabilidad total, garantizar la reversibilidad y ser consistentes con las reglas de evolución cuánticas. No obstante, esto no quiere decir que no exista la opción de construir operaciones no lineales en ordenadores cuánticos, de hecho hay varias maneras de hacerlo tanto de manera coherente como de manera no coherente, [HCSS21].

Barren Plateau

Como ya se comentó, el circuito se entrena mediante un proceso de optimización en el que se van actualizando sus parámetros. El elemento clave en dicha actualización es el gradiente de la función de coste en un punto dado, el cual determina la dirección y la magnitud del ajuste. Las *barren plateaus* es un fenómeno que puede ocurrir durante el proceso de entrenamiento de los VCs. Se produce cuando los gradiente de la función de coste se encuentran en un

entorno cercano al cero en la mayoría de los elementos del espacio paramétrico. Esto dificulta el ajuste eficiente de los parámetros desencadenando una lenta convergencia y en el peor de los casos imposibilitando el entrenamiento.

En [MBS+18] Jarrod R. McClean, Ryan Babbush junto con un grupo de investigadores estudiaron, tanto analítica como numéricamente, para una amplia clase de circuitos cuánticos aleatorios, que los valores de los gradientes se concentraban en un entorno del cero en el proceso de entrenamiento. Llegando a la conclusión de que los circuitos inicializados aleatoriamente con una profundidad suficiente carecen de utilidad dentro de la gama de algoritmos híbridos cuántico-clásicos. En las gráficas 2.5 y 2.6 se pueden observar parte de los resultados de los experimentos, en donde muestran un decaimiento exponencial de la varianza en función del número de qubits y de la profundidad del circuito. Un enfoque que proponen es realizar un entrenamiento segmento a segmento o usar aproximaciones iniciales ya estructuradas.

En general, las barren plateaus aparecen cuando los circuitos son profundos y complejos, es decir, con muchas capas, muchos qubits y muchas conexiones entre ellos. Este fenómeno se debe a la estructura de las operacionales cuánticas usadas en estos circuitos, ya que por ejemplo, las rotaciones pueden generar gradientes pequeños.

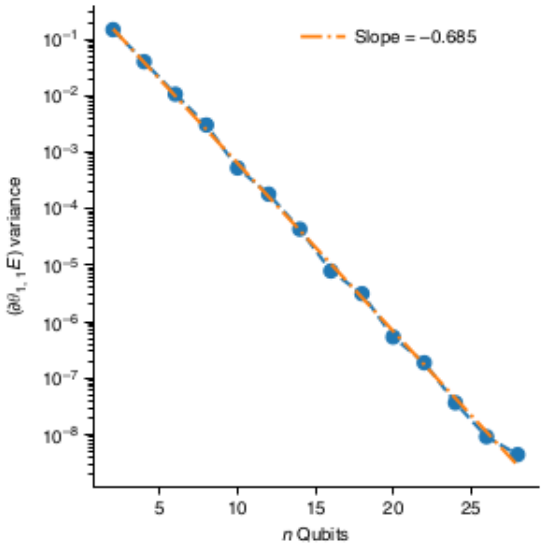


Figura 2.5: Gráfica en la que se aprecia el decaimiento exponencial de la varianza en función del número de qubits. Fuente [MBS+18]

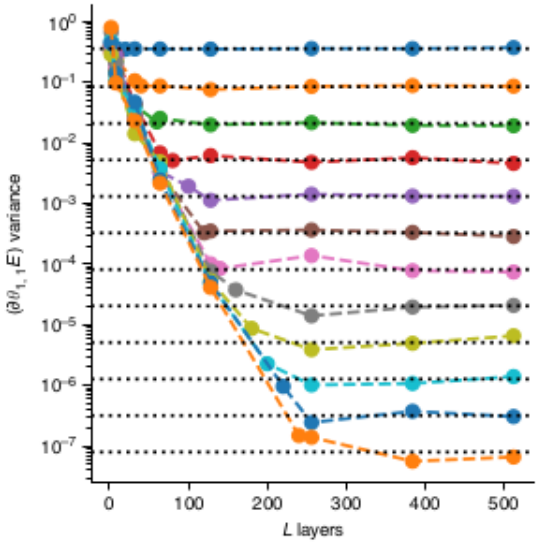


Figura 2.6: En esta gráfica se observa el decaimiento de la varianza en función de la profundidad del circuito. Las líneas se corresponden con un número distinto de qubits, desde 2 a 24 qubits de manera ordenada. Fuente [MBS+18]

3 Codificación de datos

La computación cuántica se enfrenta en la actualidad a una serie de adversidades que están impidiendo explotar su verdadero potencial. Éstas son tanto hardware como software. Por una parte, este nuevo paradigma de computación ha provocado un cambio disruptivo a la hora del desarrollo software, pues hay que cambiar la filosofía tanto del diseño como de la programación de los algoritmos. Por otra parte, el desarrollo hardware discurre más lento que el software, obligando a este último a adaptarse a las capacidades que pueden ofrecer. En esta sección se tratará uno de los principales problemas software, limitado también por el hardware actual, la codificación de datos clásicos en estados cuánticos. En la sección 3.1 se dará una introducción general a algunos problemas que la computación cuántica presenta hoy en día, y en concreto, se introducirá el problema de la codificación de datos. En la sección 3.2, se dará una breve introducción al concepto de los mapas de características cuánticos (Quantum Feature Map), y las secciones ,3.3, 3.4, 3.5, 3.6 y 3.7 tratarán de las codificaciones *Bases Encoding*, *Amplitude Encoding*, *Angle Encoding*, *Qsample Encoding* y otras codificaciones usando ciertos mapas de características, respectivamente.

3.1. Introducción

La computación cuántica se enfrenta a día de hoy a una serie de desafíos que limitan la explotación de su verdadero potencial y son el motivo fundamental por el cual los ordenadores cuánticos actuales aún no son puramente cuánticos. La principal limitación es la tecnología, de la cual derivan la mayor parte de los problemas. Uno de los problemas más famosos es el ruido. Actualmente predominan los chips de superconducción y empresas pioneras como IBM apuestan vehementemente por esta tecnología debido a su gran escalabilidad y a que por una parte, son potencialmente fáciles de fabricar en comparación con otras tecnologías, como por ejemplo, los circuitos fotónicos o las trampas de iones. Sin embargo, esta tecnología presenta algunos inconvenientes, como por ejemplo, los pequeños tiempos de coherencia, la escasa interconexión de

qubits, lo que supone un problema de cara al entrelazamiento además de la alta tasa de ruido que presentan. No obstante, hay mucho estudio detrás de la corrección de errores y actualmente se aplican sofisticadas técnicas para mitigarlos.

También existen otros problemas como la refrigeración, ya que un ordenador cuántico construido con chips superconductores necesita estar a una temperatura cercana del cero absoluto, lo cual no es fácil. Sin embargo, esto se ha podido subsanar gracias a la tecnología de refrigeración por dilución. Otras tecnologías como por ejemplo las trampas de iones, los circuitos fotónicos o los qubtis topológicos no presentan el ruido como uno de sus principales problema. En cambio, el desafío que presentan es mucho más complejo ya que requieren de una tecnología que aún está por desarrollar.

Otro gran problema que presenta la computación cuántica y el cual es independiente del hardware, a diferencia del ruido, es la codificación de datos clásicos en sistemas cuánticos. La física clásica y la cuántica son dos teorías diferentes que tratan de describir y comprender la naturaleza. La primera se encarga de explicar el comportamiento del universo a escala macroscópica, mientras que la segunda lo hace a escala microscópica. De manera aislada, cada una tienen sentido y no se contradicen a sí mismas. En cambio, al considerarlas de manera conjunta, pueden parecer contradictorias, surgiendo conflictos que entran en el marco del problema de la correspondencia, el cual es un tema actual de investigación y debate en la comunidad científica. El objetivo de esta reflexión es entender que no es fácil combinar ambas teorías. En un contexto computacional, ocurre que los qubits tienen propiedades que no tienen los bits, por lo que es un factor clave a tener en cuenta a la hora de la codificación. Además, los qubits tienen una mayor sensibilidad a errores, por lo que la información es más susceptible a ser alterada que en los bits. Por otra parte, y en lo que respecta a la capacidad hardware, ocurre que un ordenador cuántico no dispone de la misma cantidad de qubits que de bits dispone uno clásico. Toda la información que es capaz de albergar un Gigabyte resulta imposible codificarla en un ordenador cuántico actual. Con todo esto en mente uno ya puede intuir que la codificación de datos clásicos en sistemas cuánticos no es algo trivial. Hay diferentes maneras de codificar información en un sistema de n-qubits descrito por un estado, y en esta sección se verán algunas estrategias relevantes.

3.2. Quantum Feature Maps

Un mapa de características o *feature map* es una aplicación, generalmente no lineal, que mapea los datos clásicos de entrada a un espacio de Hilbert cuántico, es decir, transforma los datos a una representación cuántica adecuada.

La elección del mapa de características es de vital importancia, ya que determina por una parte cómo se representan los datos en el espacio cuántico y por otra puede afectar el rendimiento y la capacidad de clasificación del algoritmo cuántico. Estos mapas de características son comúnmente circuitos cuánticos parametrizados, un conjunto de puertas que operan sobre los qubits con la finalidad de obtener el estado cuántico deseado.

Los mapas de características son muy usados en el aprendizaje automático cuántico. Para obtener ventaja cuántica sobre los algoritmos de QML es importante diseñar circuitos que sean difíciles de simular clásicamente. Esto es un área actual de investigación.

3.3. Basis Encoding

Este tipo de codificación asocia una cadena de N bits a un estado la base computacional de un sistema de N qubtis. Esta es la manera más directa de codificación ya que cada bit se sustituye literalmente por un qubit y el cálculo se realiza en superposición.

Sea

$$S = \{x_i | x_i \in \{0,1\}^N \ , \ i = 1, ..., M\}$$

un dataset que contiene M cadenas de bits de longitud N. Teniendo en cuenta que una cadena de N bits $x \in \{0,1\}^N$ se corresponde con un estado de la base computacional de un sistema de N qubits, $|x\rangle$ se puede representar el dataset entero como una superposición de estados de la base computacional, esto es,

$$|S\rangle = \frac{1}{\sqrt{M}} \sum_{i=1}^{M} |x_i\rangle$$

Ejemplo 3.1. Sea $S = \{101, 111\}$ un dataset de dos elementos de longitud tres, por lo que $M = 2$ y $N = 3$. La representación de S como superposición de elementos de la base computacional quedaría,

$$|S\rangle = \frac{1}{\sqrt{2}}\Big(|101\rangle + |111\rangle\Big)$$

el vector de amplitud de S en este caso es $\alpha = (0,0,0,0,0,\frac{1}{\sqrt{2}},0,\frac{1}{\sqrt{2}})^T$

Este tipo de codificación es bastante simple. Sin embargo, está limitada a una cierta representación de datos, como por ejemplo números enteros o binarios. Aunque también se puede aplicar a números reales haciendo una conversión y teniendo en cuenta siempre errores de imprecisión, [LC20]. Por ejemplo, para el vector $x = (0.1, -0.6, 1.0)$ se puede representar en binario usando un primer bit para indicar el signo y tomando los 4 bits siguientes de precisión decimal, de este modo $x = (00001, 11001, 01111)$. Hay que tener en cuenta también que por otra parte es ineficiente ya que el número de qubits a usar escala exponencialmente.

Aunque no se entre en detalle, comentar que en [VM00] y [Tru01] se propone una manera elegante de construir esta superposiciones de estados en tiempo linear en M y en N.

3.4. Amplitude Encoding

Según [LC20] esta codificación es mucho menos común en computación cuántica, aunque dentro del área del quantum machine learning es bastante usada. Asocia un vector clásico de variable compleja con amplitudes cuánticas y hay varias maneras de llevarlo a cabo aunque aquí solo se verá la más común.

Sea $n \in \mathbb{N}$ una cantidad arbitraria de qubtis y sea $x \in \mathbb{C}^{2^n}$ un vector normalizado, es decir, verificando $||x|| = \sum_{i=1}^{2^n} |x_i|^2 = 1$. Este tipo de codificación representa a x por medio de las amplitudes de un estado cuántico $|\psi\rangle \in \mathcal{H}$ vía

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_{2^n} \end{bmatrix} \leftrightarrow |\psi_x\rangle = \sum_{i=1}^{2^n} x_i |i\rangle$$

Análogamente, dada una matriz $A \in \mathbb{C}^{2^n \times 2^m}$ unitaria con $a_{ij} \in \mathbb{C}$ sus

elementos, se puede codificar como

$$|\psi_A\rangle = \sum_{i=1}^{2^n} \sum_{j=1}^{2^m} a_{ij}|i\rangle|j\rangle$$

Esta codificación es bastante útil ya que los índices del vector se corresponden con los estados $|i\rangle$ de la función de onda y el número de qubits a usar para la codificación escala de manera logarítmica con respecto a la cantidad de datos a codificar. Sin embargo, codificar la información a amplitudes de probabilidad de una función de onda presenta muchas limitaciones sobre las operaciones que se pueden ejecutar. La mayor limitación radica en que si se considera una función no lineal f, entonces la codificación de $f(x)$ en un estado cuántico $|\psi_{f(x)}\rangle$ resultaría imposible partiendo de $|\psi_x\rangle$, [Per89, AL98]. Otra limitación es que en algunos casos se mete información redundante, esto es, si se quiere codificar un vector tridimensional $x = (x_1, x_2, x_3) \in \mathbb{C}^3$, se necesitaría un sistema de 2 qubits, donde $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{C}^{2^2}$ es el vector de amplitudes de probabilidad. Por lo que se tendría que $x_i = \alpha_i$ para $i = 1, 2, 3$ y $\alpha_4 = 0$. El vector x se codifica en un estado cuántico con un grado de libertad menos ya que el dominio de x sería un hiperplano (esfera tridimensional) en un espacio de dimensión 2^2. Una alternativa es hacer padding en el vector x y añadirle tantas componentes con valor 1 como se necesite.

Ejemplo 3.2. Sea $x = (0.1, -0.6, 1.0) \in \mathbb{R}^3$. El número de qubits necesario para codificar x es $\lfloor log_2(3) \rfloor + 1 = 2$, por lo que el vector será codificado en un estado cuántico de dos qubits. Primeramente se normaliza el vector y posteriormente se añaden tantas componentes nulas como amplitudes tenga el estado sobre el que se va a codificar (también se podría inicializar la última componente a uno y normalizar), en este caso el vector quedaría como

$$x' = (0.073, -0.438, 0.730, 0.000)$$

Ahora, la representación en un estado de 2 qubits quedaría como

$$|\psi_x\rangle = 0.073|00\rangle - 0.438|01\rangle + 0.730|10\rangle + 0|11\rangle$$

Notar que este estado también podría ser codificado como una matriz de la siguientes manera

$$\begin{bmatrix} 0.073 & -0.438 \\ 0.730 & 0.000 \end{bmatrix}$$

Como antes se comentó esta codificación es bastante usada en los algoritmos de quantum machine learning. Más concretamente, un dataset S se puede ver como una matriz en $\mathbb{C}^{N\times M}$ siendo M el número de instancias y N el número de características o la dimensionalidad de cada instancia. Dicho lo cual, la codificación quedaría como

$$|\psi_S\rangle = \sum_{i=1}^{M}\sum_{j=1}^{N} x_j^i |j\rangle|i\rangle = \sum_{i=1}^{M} |\psi_{x^i}\rangle|i\rangle$$

Este estado cuántico tiene un vector de amplitud de dimensión NM y es construido concatenando todos las instancias del dataset,

$$\alpha = (x_1^1, ..., x_N^1, \ldots, x_1^M, \ldots, x_N^M)$$

Las salidas del conjunto de entrenamiento pueden ser también codificadas o bien en qubits extras $|y^i\rangle$ entrelazados con los respectivos registros $|i\rangle$ con $i = 1, \ldots, M$ o bien codificándolas como amplitudes de un estado cuántico en un registro separado,

$$|\psi_y\rangle = \sum_{i=1}^{M} y^i |i\rangle$$

En líneas generales, esta técnica de codificación de conjuntos de datos requiere la preparación de un estado arbitrario

$$|\psi\rangle = \sum_i \alpha_i |i\rangle$$

de manera robusta y eficiente. De hecho, la preparación del estado se puede hacer en tiempo lineal siguiendo una rutina propuesta por Möttönen et al. [MVBS04].

Por último hacer hincapié en que si se han de codificar NM amplitudes y teniendo en cuenta que un sistema de n qubits posee un total de 2^n amplitudes, basta con que se cumpla $n \geq \log_2(NM)$.

Ejemplo 3.3. Sea $S = x^1 = (2, 0), x^2 = (-2, 10)$ un dataset. Para codificarlo usando este método primero se ha de tener en cuenta que $M = 2$ y $N = 2$ por lo que el número de qubtis a usar es $n = \log_2(2^2) = 2$. A continuación se considera el vector

$$\alpha' = (2, 0, -2, 10)$$

obtenido tras concatenar x^1 con x^2, tras normalizarlo quedaría

$$\alpha = \frac{1}{\sqrt{108}}\alpha' = \frac{1}{\sqrt{108}}(2, 0, -2, 10)$$

Finalmente se asociada cada elemento del vector con un estado de la base computacional resultando

$$|\psi_S\rangle = \frac{1}{\sqrt{108}}(2|00\rangle - 2|10\rangle + 10|11\rangle)$$

3.5. Angle Encoding

Según [bai, Pen] este tipo de codificación es el camino más directo para codificar información clásica en estados cuánticos, y resulta muy interesante sobre todo en el contexto de las redes neuronales cuánticas y en las redes tensoriales, *tensor networks*.

Este método codifica una característica por qubit, al contrario de los dos anteriores, que podían codificar el dataset entero. Por tanto si se tienen N características será necesario un sistema con un número de qubtis n que cumpla $N \leq n$. Para realizar esta codificación es necesario mapear el vector al intervalo $[0, 2\pi[$. Dado un vector $x = (x_1, \ldots, x_N) \in [0, 2\pi[$, quedaría como

$$|x\rangle = \bigotimes_{i=1}^{N} cos(x_i)|0\rangle + sin(x_i)|1\rangle$$

Esta codificación se puede formular como una matriz unitaria de la forma

$$O_{x_j} = \bigotimes_{i=1}^{N} \begin{bmatrix} cos(x_j^i) & -sin(x_j^i) \\ sin(x_j^i) & cos(x_j^i) \end{bmatrix}$$

Recalcar que una rotación de un determinado ángulo θ en el eje Y corresponde a un giro en el plano X-Z y la expresión matricial viene dada por

$$RY(\theta) = e^{-i\frac{\theta}{2}Y} = \begin{bmatrix} cos(\frac{\theta}{2}) & -sin(\frac{\theta}{2}) \\ sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{bmatrix}$$

por tanto se tiene que,

$$O_{x_j} = \bigotimes_{i=1}^{N} RY(2x_j^i)$$

De hecho, esta codificación también admite rotaciones RX y RZ.

En el ámbito de la redes neuronales cuánticas (*quantum neural networks*), se suele introducir la entrada de una red $\Theta = w_0 + w_1 x_1 + \dots x_N x_N$ en el ángulo de un qubti ancilla y se entrelaza con la salida de algún estado arbitrario $|\psi\rangle$, quedando

$$RY(2\theta)|0\rangle \otimes |\psi\rangle_{out} = (cos(\theta)|0\rangle + sin(\theta)|1\rangle \otimes |\psi\rangle_{out}$$

El qubti de la entrada de la red neuronal se rota un ángulo θ a lo largo del eje Y. Ahora lo interesante sería poder aplicar al qubit de salida una rotación dada por una función de activación no linear ϕ que dependa de θ, es decir, preparar el siguiente estado $RY(2\phi(\theta))|\psi\rangle$. Una manera muy elegante de conseguir lo anterior para funciones sigmoidales viene dada por Cao et al. [CGAG17].

3.5.1. Dense Angle Encoding & General Qubit Encoding

Esta codificación es una generalización de la anterior en la que se pueden codificar dos características por qubit. Un ejemplo de como quedaría puede verse en la figura 3.1

Definición 3.1 (Dense Angle Encoding, [LC20])**.** Dado un vector de características $x \in \mathbb{N}$, la codificación *dense angle encoding* aplica la asignación $x \to E(x)$ dada por

$$|x\rangle = \bigotimes_{i=1}^{\lceil N/2 \rceil} cos(\pi x_{2_i - 1})|0\rangle + e^{2\pi i x_{2i}} sin(\pi x_{2i-1})|1\rangle$$

Esta codificación para elementos de dos dimensiones $x \in \mathbb{R}^2$ realizada con un único qubit viene dada por

$$|x\rangle = cos(\pi x_{2_1})|0\rangle + e^{2\pi i x_2} sin(\pi x_1)|1\rangle$$

y tiene matriz de densidad

$$\rho_x = \begin{bmatrix} cos^2(\pi x_1) & e^{-2\pi i x_2} cos(\pi x_1) sin(\pi x_1) \\ e^{2\pi i x_2} cos(\pi x_1) sin(\pi x_1) & sin^2(\pi x_1) \end{bmatrix}$$

Aunque tanto las codificaciones *Angle Encoding* y *Dense Angle Encoding* se hagan con funciones sinusoidales, no quiere decir que no se puedan hacer con otro tipo de funciones. De hecho, se puede lograr mayor abstracción y definir una codificación mucho más general con funciones arbitrarias.

Definición 3.2 (General Qubit Encoding, [LC20])**.** Dado un vector de características $x \in \mathbb{N}$, la codificación *General Qubit Encoding* aplica la siguiente asignación $x \to E(x)$ dada por

$$|x\rangle = \bigotimes_{i=1}^{\lceil N/2 \rceil} f_i(x_{2_i-1}, x_{2i})|0\rangle + g_i(x_{2i-1}, x_{2i})|1\rangle$$

donde $f_i, g_i : \mathbb{R} \times \mathbb{R} \to \mathbb{C}$ cumplen $||f_i||^2 + ||g_i||^2 = 1$ para todo $i = 1, \ldots, N$

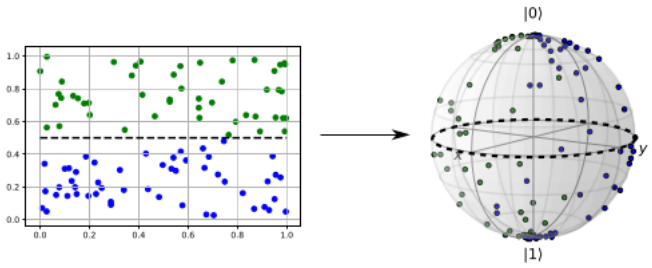


Figura 3.1: Una representación visual de aplicar la codificación *dense angle encoding* a un conjunto de puntos generador aleatoriamente y normalizados separados por una frontera. Imagen obtenida de [LC20]

3.6. Qsample Encoding

Se considera un sistema de n qubits que describe un estado con un vector de amplitud $\alpha = (\alpha_1, \ldots, \alpha_2^n)^T$. Medir los n qubits de la base computacional equivale a muestrear una cadena de bits de longitud n a partir de una distribución de probabilidad discreta $p_1 = |\alpha_1|^2, \ldots, p_{2^n} = |\alpha_{2^n}|^2$. Por *equivalente*, se entiende que, tanto la medición como la rutina de muestreo clásica extraen muestras con la misma probabilidad. Lo que significa que se puede utilizar el vector amplitud para representar una distribución de probabilidad discreta

clásica. Dada una distribución de probabilidad discreta clásica sobre cadenas binarias $p_1, \ldots, p_{2^n}$, el estado cuántico viene dado por

$$|\psi\rangle = \sum_{i=1}^{2^n} \sqrt{p_i}|i\rangle$$

Esta codificación es conocido como *qsample*, [ROAG17].

3.7. Otras codificaciones por medio de Features Maps

Hay muchos estudios sobre mapas de características que son usados para la codificación de datos clásicos en estados cuánticos. En esta sección se verán algunos de los más conocidos.

El primero de ellos es conocido con el nombre de *Pauli Feature Map*, y es propuesto en [HCT+19] e implementado por Quiskit [Qui20a]. El mapa de características con profundidad $d \in \mathbb{N}$ viene dado por el siguiente operador unitario

$$\mathcal{U}_{\Phi(x)} = \prod_d U_{\Phi(x)} H^{\otimes n}$$

donde

$$U_{\Phi(x)} = \exp\left(i \sum_{L \subseteq [n]} \phi_L(x) \prod_{k \in L} P_i\right)$$

y donde, $n \in \mathbb{N}$ es el número de qubtis, L es un conjunto de índices de qubits que describen las conexiones en el mapa de características, $[n] = \{\binom{N}{k}, k = 1, ..., n\}$ es un conjunto conteniendo todos los conjuntos índices, $P_i \in \{I, X, Y, Z\}$ y

$$\phi_L(x) = \begin{cases} x_i & \text{si} \quad L = \{i\} \\ \prod_{j \in L}(\pi - x_j) & \text{si} \quad |L| > 1 \end{cases}$$

un caso particular de este mapa de características es cuando $k = 2$, $P_0 = Z, P_1 = ZZ$, y es conocido como *ZZFeatureMap* también implementado en Quiskit [Qui20b] y el cual viene dado por la siguiente expresión

$$\mathcal{U}_{\Phi(x)} = \left(\exp\left(i \sum_{jk} \phi_{\{j,k\}}(x) Z_j \otimes Z_k \right) \exp\left(i \sum_{j} \phi_{\{j\}}(x) Z_j \right) H^{\otimes n} \right)^d$$

4 Redes Adversarias Generativas Cuánticas - QGANS

El aprendizaje automático cuántico es actualmente una de las apuestas más prometedoras en el mundo de la computación cuántica, [KKP21]. Según Jungsang Kim, uno de los fundadores y directores de IonQ, los modelos cuánticos de aprendizaje automático son mucho más expresivos que los clásicos gracias a que son capaces de capturar y procesar patrones realmente complejos en los datos, que de otra manera, resultaría muy complicado hacer. Dentro del aprendizaje automático, las redes adversarias generativas han logrado importantes avances gracias a su potencial, [GPAM+14]. En este capítulo se presentará la versión cuántica de estos modelos y como son modelizables usando circuitos variacionales. El capítulo está estructurado de la siguiente manera: en la sección 4.1 se verá una pequeña introducción, en la sección 4.2 se introducirá el concepto de QGAN junto con su modelización en un circuito variacional y por último en la sección 4.3 se discutirá el problema de optimización usado en el entrenamiento del modelo así como su función de coste.

> La computación cuántica cambiará fundamentalmente la forma en que resolvemos problemas y abrimos nuevas posibilidades en múltiples campos científicos.
>
> *(Peter Shor)*

4.1. Introducción

La computación cuántica, una tecnología basada en las propiedades de los sistemas cuánticos, es un tema candente de investigación que se ha ido popularizando en las últimas dos décadas, [McK22]. El desarrollo de esta tecnología viene motivado por la suposición de que será capaz de afrontar y resolver de manera eficiente problemas importantes y clásicamente intratables. De he-

cho, a día de hoy no hay ninguna demostración rigurosa que exponga que la computación cuántica sea capaz de resolver problemas que clásicamente sean intratables. No obstante, sí que existen algoritmos cuánticos que muestran un rendimiento superior a los clásicos en algunos tipos de problemas, como por ejemplo, el algoritmo de factorización de Shor, [Sho99]. De hecho, Ryan Babbush y un equipo de investigadores demostraron en [BMN+21] que para que un modesto ordenador cuántico tolerante a fallos - recalcar que a día de hoy no existen, es una tecnología futura - logre una ventaja de tiempo de ejecución sobre un enfoque clásico, el algoritmo cuántico debería ofrecer un aumento de velocidad de al menos un grado cuadrático. También expone que dicho aumento cuadrático sería lo mínimo indispensable para lograr dicho fin aunque no sería suficiente para poder aprovechar la ventaja cuántica de cara a las primeras generaciones de los ordenadores. Tras algunos experimentos llega a la conclusión de que los aumentos de velocidad cuárticos parecen significativamente más prácticos.

Los computadores cuánticos actuales son llamados NISQ (*Noisy Intermediate-Scale Quantum*). Estos ordenadores no están libres de ruido - de hecho esa es su principal limitación - y todavía están lejos de ser capaces de realizar cálculos cuánticos completamente fiables y escalables. Los principales problemas de los NISQ son el ruido, los tiempos de coherencia y la conectividad de los qubits, siendo este último factor crucial de cara al entrelazamiento. A día de hoy ni los NISQ ni incluso los *quantum annealers* de Dwave han logrado superar de manera consistente a los ordenadores clásico en términos de tiempo de ejecución y calidad de la solución, por lo cual no no se ha alcanzado aún la supremacía cuántica. Pese a esto, los ordenadores cuánticos son los mejores candidatos en lo que respecta la simulación de sistemas inherentemente cuánticos, como por ejemplo, interacciones de partículas o reacciones químicas.

En la actualidad existe un gran afán de explotar las tecnologías cuánticas presentes. Debido a esto y teniendo siempre en cuenta los factores limitantes, han surgido algoritmos híbridos cuántico-clásicos (HQC) - algunos ejemplos de HQC incluyen el VQE (*Variational Quantum Eigensolver*) [PMS+14], QVECTOR (Variational Quantum Error Correction Scheme) [JRO+17], QAE (Quantum Autoencoder) [ROAG17], entre otros - con el propósito de ser usados en los dispositivos cuánticos actuales y en los a corto-medio plazo. De este modo, unas series de tareas serán ejecutadas en un dispositivo cuántico mientras que otras serán tratadas de manera clásica. Existe un subconjunto dentro de estos algoritmos, llamado AHQC (*Adaptative Hybrid Quantum-Classical*) en los que la parte cuántica se encargan de preparar un estado por medio de un circuito

cuántico variacional 2 midiendo después el valor esperado de los qubtis de salida que proporcionarán la información a usar en la siguiente fase. La parte adaptativa hace referencia a que los parámetros de los circuitos cuánticos se ajustan dinámicamente durante el proceso de ejecución del algoritmo. Son usados actualmente en problemas de optimización, simulación de materiales y aprendizaje automático.

Por otra parte, existen numerosas líneas de investigación, algunas de ellas abordan la conexión entre las redes neuronales y los circuitos variacionales. Uno de los problemas más difíciles en la actualidad es la codificación de datos clásicos en un ordenador cuántico, es decir, la cuantización de datos que provengan de fuentes clásicas. Existen algunas estrategias para esto, pero en términos generales son bastantes ineficientes por requerir de una gran cantidad de qubtis. Para más detalle sobre esta discusión ir a 3

En aprendizaje automático, los modelos discriminadores o clasificadores son entrenados para aprender una distribución de probabilidad $p(y|x)$ de la variable etiqueta y condicionada a un conjunto de observaciones x. Por el contrario, los modelos generativos se entrenan para aprender la distribución de probabilidad conjunta $p(x, y)$ o alternativamente, la distribución de probabilidad condicionada de una muestra observada con respecto a la etiqueta, $p(x|y)$. La mayoría de los algoritmos HQC para modelos clasificadores usan clasificadores cuánticos variacionales, donde se optimizan los parámetros de un circuito variacional para aprender directamente la distribución $p(x|y)$. Otra estrategia usada para estos problemas son las máquinas de soporte vectorial cuánticas. Mientras tanto, los algoritmos HQC para tareas generativas se han centrado en trabajar con distribuciones discretas usando circuitos variacionales, en particular una QCB (Quantum Born Machine). Los problemas que involucren distribuciones continuas se han estudiado menos debido a su dificultad pero actualmente se está comenzando a abordar. Ejemplos de estos problemas pueden ser la generación de sonidos o imágenes sintéticas. Los principales problemas que presentan estas tareas generativas enfocadas en el marco de la computación cuántica son dos. El primero es la codificación de datos clásicos en un estado cuántico de manera eficiente. Y el segundo es el número de mediciones requeridas para poder muestrear la distribución aprendida, que crece exponencialmente.

4.2. QGANs

Una red adversaria generativa cuántica es muy similar a su homóloga clásica ya que siguen compartiendo la misma estructura y filosofía como puede observarse en la imagen 4.3. Las principales diferencias radican en que se sustituyen las redes neuronales del generador y del discriminador por circuitos variacionales y el espacio latente muestrea estados cuánticos en vez de vectores aletorios. Las QGANs no tiene por qué ser puramente cuánticas, en la imagen 4.2 se puede apreciar las distintas combinaciones que se pueden hacer en función de la naturaleza de los datos, generador y discriminador. Es muy complicado explotar el potencial de una QGAN debido a las limitaciones hardwares actuales de los ordenadores cuánticos. Debido a ello, se suelen priorizar o bien el discriminador, o bien el generador. El esquema más común de todos es cuando solo el generador está cuantizado, pues al ser quien se encarga de aproximar la distribución de probabilidad es la parte más crítica. A continuación se detallará el comportamiento de una QGAN púramente cuántica por ser el caso más general. No obstante, partiendo de esto resulta bastante sencillo particularizar a casos híbridos.

La estructura de una QGAN totalmente cuántica viene reflejada en la figura 4.1. Se parte de una fuente de datos R la cual opera sobre un sistema formado por tres registros de s, n y m qubits respectivamente. Dada una etiqueta λ - información adicional y opcional - el sistema genera como salida una matriz de densidad $R(|\lambda\rangle) = \rho_\lambda^R$ en el registro de n qubits. El objetivo es encontrar un generador G que imite la fuente de datos reales R. Se define G como un circuito variacional cuyas puertas están parametrizadas por un vector θ_G. De manera similar a la versión clásica, se toma un elemento $|z\rangle$, que ahora será un estado del espacio latente y que vendrá cargado en un registro de trabajo (Batch R/G) de m qubits que junto con la etiqueta $|\lambda\rangle$, cargada en el otro registro (Label R/G) de s qubits, produce un estado

$$G(\theta_G, |\lambda, z\rangle) = \rho_\lambda^G(\theta_G, z)$$

donde ρ_λ^G es la salida contenida en otro registro (Out R/G) de n qubtis, al igual que el registro de los datos reales. Sin pérdida de generalidad se asume que el registro Batch R/G es inicializado al estado $|0\rangle^{\otimes m}$ cuando se está usando la fuente de datos R en vez del generador G.

El estado del espacio latente $|z\rangle$ juega un doble rol. Por una parte puede ser entendido como una fuente con ruido no estructurado que aporta entropía

a la distribución de los datos generados. Por ejemplo, podríamos tener un generador unitario que produzca un estado fijo $\rho_{\lambda}^{G}(\theta_G, z) = |\psi_\lambda(z)\rangle\langle\psi_\lambda(z)|$ para cada λ y $|z\rangle$. Si se permite que la entrada fluctúe aleatoriamente, se puede crear más de un estado de salida para cada etiqueta. Por otra parte, el estado $|z\rangle$ puede servir como un elemento de control para el generador, tuneando el estado $|z\rangle$ podemos transformar el estado de salida preparado por G, variando algunas propiedades de los datos generados las cuales no son capturadas por λ. Mientras que el primer papel se puede lograr añadiendo un registro adicional **Bath G**, el segundo requiere que $|z\rangle$ esté bajo nuestro control, aunque no esté dotado de ninguna estructura.

El discriminador es un circuito cuántico independiente del generador parametrizado por un vector θ_D. Al igual que en la versión clásica, la salida del generador G o la de la fuente de datos R es mandada al discriminador D por medio del registro Out R/G. Apoyándose en un registro de trabajo Bath D de d qubtis y otro registro Label D de s qubtis conteniendo $|\lambda\rangle$, el discriminador determina si la entrada recibida es por parte de la fuente de datos reales R o del generador, emitiendo por el registro Out D $|\rangle$ o $|fake\rangle$ respectivamente. Por último, notar que el valor esperado del operador

$$Z \equiv |real\rangle\langle real| - |fake\rangle\langle fake|$$

del registro $OutD$ es proporcional a $Pr\Big(D(\theta_D, |\lambda\rangle), R(|\lambda\rangle) = |real\rangle\Big)$ y puede ser usado para definir el problema de optimización en un contexto más puramente de la mecánica cuántica.

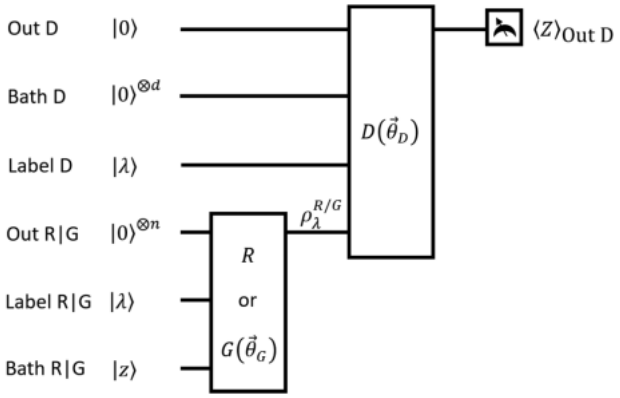


Figura 4.1: Estructura genera de una QGAN púramente cuántica. Aunque en la imagen aparece un único circuito, en realidad hay dos, uno para cuando se use el generador y otro para cuando sea la fuente de datos. Los registros *Bath* son registros de trabajo, el del discriminador consta de d qubtis y los del generador y la fuente de datos R de n qubtis. La fuente de datos reales R o el generador parametrizado $G(\theta_G)$ se aplicada sobre un estado inicial $|0^{\otimes n}, z, \lambda\rangle$, definido en los registros *Out R/G*, *Label R/G* y *Batch R/G* respectivamente, elaborando una salida $\rho_\lambda^{R/G}$. El discriminador $D(\theta_D)$ recibe la información $\rho_\lambda^{R/G}$ de entrada junto con un estado $|0, 0^{\otimes d}, \lambda\rangle$, definido por los registros *Out D* y *Batch D* respectivamente. El discriminador emite su salida $|Real\rangle$ o bien $|Fake\rangle$ en el registro *Out D*. El valor esperado $\langle Z\rangle_{Out\ D}$ es proporcional a la probabilidad de que D saque la salida $|Real\rangle$.

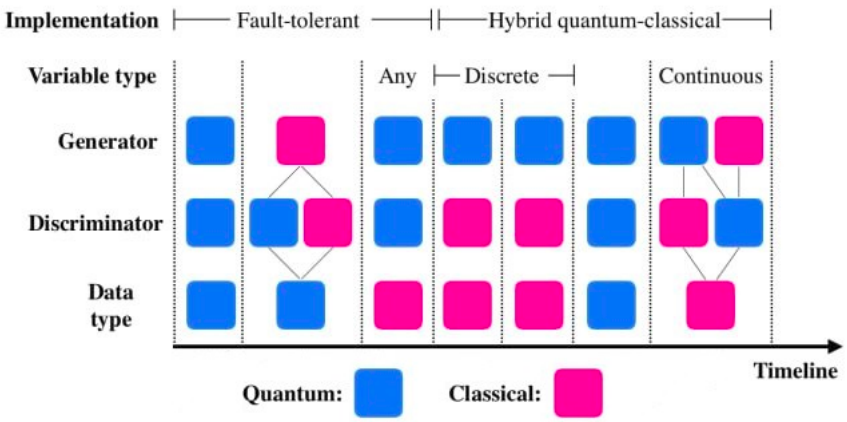


Figura 4.2: Esquema de las distintas implementaciones de QGANs. Este esquema deja una división tripartita en términos de la naturaleza de los datos, del discriminador y del generador. Si la naturaleza de los datos es clásica, se subcategorizan atendiendo a si los datos son de naturaleza continua o discreta. También se describe para qué tipo de computador cuántico están diseñadas, si uno tolerante a fallos o para un NISQ.

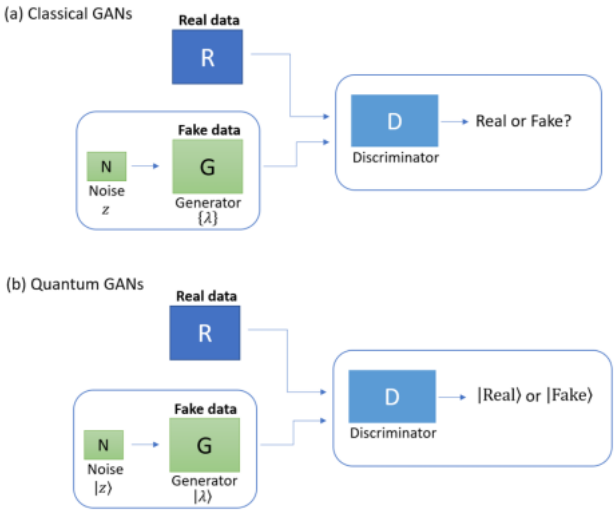


Figura 4.3: (a) Estructura de una GAN clásica. El discriminador deberá decidir si las muestras son dadas por una fuente de recursos reales R o por el generador $G(z)$, donde z es un vector aleatorio. (b) Estructura de una QGAN, donde el discriminador debe decidir si el estado cuántico recibido como entrada proviene de una fuente real R o un generador cuántico $G(|z\rangle)$. La salida del discriminador será $|Real\rangle$ o $|Fake\rangle$

4.3. Problema de optimización y función de coste

El problema de optimización planteado durante el entrenamiento viene dado por

$$min_{\theta_G} max_{\theta_D} V(\theta_D, \theta_G)$$

donde

$$V(\theta_D, \theta_G) =$$

$$\frac{1}{\Lambda} \sum_{\lambda=1}^{\Lambda} Pr\left(\Big(D(\theta_D, |\lambda\rangle, R(|\lambda\rangle)) = |real\rangle \Big) \cap \Big(D(\theta_d, |\lambda\rangle, G(\theta_G, |\lambda, z\rangle)) = |fake\rangle \Big) \right) \tag{4.1}$$

Las GANs clásicas suelen estar el logaritmo de las probabilidades para abordar este problema de optimización, pero en el caso cuántico es recomendable definir una función de coste linear para las salidas de las probabilidades de D ya que queremos optimizar una función cuya esperanza es una función lineal.

Esto se puede hacer sin pérdida de generalidad, ya que al ser el logaritmo una función estrictamente creciente, los máximos y los mínimos permanecen invariantes al componerla con otra función. En la ecuación anterior se asume que el conjunto de etiquetas Λ tiene una cardinalidad finita, aunque esto puede no ser así, aunque tampoco supone un problema ya que la ecuación puede ser modificada para dicho caso.

Se supone que el flujo del proceso de entrenamiento es el que viene detallado en la figura 4.4. Al inicio, el discriminador y el generador son inicializados por un conjunto arbitrario de parámetros θ_D^0 y θ_G^0 respectivamente. El estado de inicio del sistema es

$$\rho_\lambda^0(z) = (|0\rangle\langle 0|)^{\otimes d+1} \otimes |\lambda\rangle\langle\lambda| \otimes (|0\rangle\langle 0|)^{\otimes n} \otimes |\lambda\rangle\langle\lambda| \otimes |z\rangle\langle z|$$

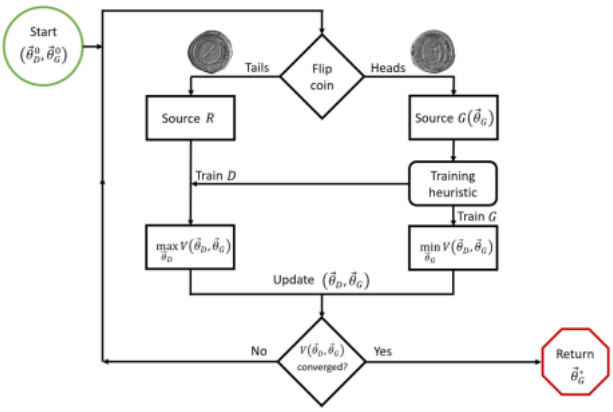


FIG. 3. Algorithmic flow of the training of a QuGAN.

Figura 4.4: Flujo del algoritmo de entrenamiento de una QGAN stándar.

El discriminador opera elemento a elemento y para impedir que haya cierto sesgo, la elección de si el elemento del entrada viene por parte de G o de R es totalmente arbitraria, como el lanzamiento de una moneda. Las operaciones unitarias correspondientes a las fuentes R y G(θ_G) y que actúan sobre todo el sistema son

$$U_R = I^{\otimes(1+d+s)} \otimes R,$$
$$U_G(\theta_G) = I^{\otimes(1+d+s)} \otimes G(\theta_G)$$

Después, tras aplicar R o G, el estado del sistema queda como

$$\rho_\lambda^R = U_R\ \rho_\lambda^0(0)\ U_R^\dagger,$$
$$\rho_\lambda^G(\theta_g, z) = U_G(\theta_G)\ \rho_\lambda^0(z) U_G^\dagger(\theta_G)$$

Ahora, la operación unitaria que define el discriminador $D(\theta_D)$ tiene la forma

$$U_D(\theta_D) = D(\theta_D) \otimes I^{\otimes m}$$

Ahora, el estado que tiene el sistema cuando se aplica $U_D(\theta_D)$ tras la fuente de datos U_R viene dado por

$$\rho_\lambda^{DR}(\theta_D) = U_D(\theta_D)\lambda_\lambda^R U_D(\theta_D)$$

y el estado cuando se aplica $U_D(\theta_D)$ después de $U_G(\theta_G)$ es

$$\rho_\lambda^{DG}(\theta_D, \theta_G, z) = U_D(\theta_D)\rho_\lambda^G(\theta_G, z) U_D^\dagger(\theta_D)$$

A través de una serie de pasos y usando algunas propiedades importantes como la propiedad de la traza cíclica para la reordenación de términos y el uso de trazas parciales para calcular el valor esperado de los observables, la función de coste reflejada en (4.1) se puede reescribir como

$$V(\theta_D, \theta_G) = \frac{1}{2} + \frac{1}{2\Lambda} \sum_{\lambda=1}^{\Lambda} \left[cos^2(\phi) tr\Big(Z\rho_\lambda^{DR}(\theta_D)\Big) - sin^2(\phi) tr\Big(Z\rho_\lambda^{DG}(\theta_D, \theta_G, z)\Big) \right]$$

donde se puede observar los términos que acompañan a la función coseno y seno dependen de θ_D mientras que solo el segundo depende de θ_G. El ángulo ϕ representa el sesgo introducido por el lanzamiento de la moneda anteriormente comentado (4.4). Suponiendo que la decisión de la moneda es equiprobable, se toma $\phi = \frac{\pi}{4}$ y el problema de optimización final quedaría como

$$min_{\theta_G} max_{\theta_D} \frac{1}{2} + \frac{1}{4\Lambda} + \sum_{\lambda=1}^{\Lambda} tr\Big(\Big[\rho_\lambda^{DR}(\theta_D) - \rho_\lambda^{DG}(\theta_D, \theta_G, z)\Big] Z\Big)$$

Por último comentar que el circuito de la figura 4.1 se puede entrenar usando el famoso método del gradiente descendente. Solo se ha de tener en cuenta qué es lo que se está entrenando en la k-ésima iteración,

$$\theta_D^{k+1} = \theta_D^k + \mu_D^k \nabla_{\theta_D} V(\theta_D^k, \theta_G^k),$$
$$\theta_G^{k+1} = \theta_G^k - \mu_G^k \nabla_{\theta_G} V(\theta_D^k, \theta_G^k)$$

5 Caso Práctico

En este capítulo se presenta un caso práctico en el que se aplican distintas arquitecturas de QGANs para resolver un problema. La sección 5.1 presenta una introducción al problema explicando muy brevemente, y sin entrar en detalle, de qué trata y qué arquitecturas se han usado. La sección 5.2 explica las arquitecturas empleadas elemento por elemento y más en detalle. En la sección 5.3 se explica la metodología de la implementación. Después, la sección 5.4 trata sobre los resultados obtenidos y el análisis de los mismos. Tras ello, la sección 5.5 expone las conclusiones y finalmente, en la sección 5.6 se comenta posibles trabajos futuros a realizar.

5.1. Introducción

En este capítulo se presentará un caso práctico, en el que se aplican varias arquitecturas de QGANs con generador y discriminador cuánticos usando 3 qubits para intentar aprender una serie de distribuciones bien conocidas. La motivación reside en que actualmente ni las simulaciones ni los ordenadores NISQ pueden ofrecer la potencia necesaria como para que las QGANs igualen a sus contrapartes clásicas, sin contar otros tipos de dificultades ya mencionadas en capítulos atrás como las *barrens plateau*. Por lo que se pretenderá demostrar que estos modelos de aprendizaje automático cuántico son capaces de capturar los patrones de otras distribuciones más sencillas, y reproducirlos.

Para aprender estas distribuciones se han usado tres tipos de arquitecturas. En todas ellas tanto el discriminador como el espacio latente se mantienen fijos y lo que varía es la arquitectura del generador. Estas arquitecturas son por una parte los circuitos parametrizados conocidos como *TwoLocal* y *EfficientSU2* y por otra la misma arquitectura que presenta el discriminador. Algunas investigaciones [JKA21, ZZM21] afirman que los circuitos *TwoLocal* y *EfficientSU2* son buenos candidatos para ser usados como arquitecturas base, de ahí la elección. También se consideró usar la misma arquitectura tanto en el discriminador como el generador para estudiar qué es lo que ocurre en igualdad de condiciones.

La experimentación ha sido programada en el lenguaje de programación *Python* y ejecutada en un ordenador Lenovo en una arquitectura x86_64 con un procesador Intel(R) Core(TM) i7-1065G7 CPU 1.30GHz. Las librerías empleadas han sido *qiskit, numpy, pandas, tensorflow, pickle* y *Matplotlib*.

5.2. Creación de los modelos

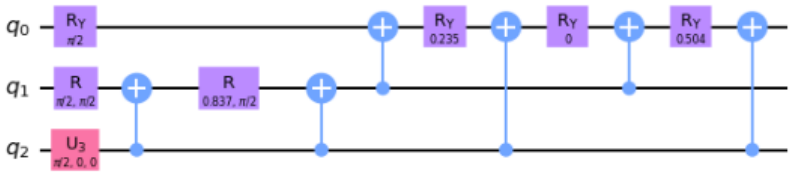
En esta sección se discutirán las arquitecturas de los tres modelos que se han usado en el experimento. En el subapartado 5.2.1 se explica los circuitos usados para cargar la distribución real de los datos. En 5.2.2 se detalla el espacio latente y la arquitectura del discriminador empleada. Ambos elementos resultan ser los mismas en todas las arquitecturas. Y en el último subapartado 5.2.3, se explican las distintas estructuras de los generadores, siendo los elementos que diferencian los modelos.

5.2.1. Circuitos de Datos Reales

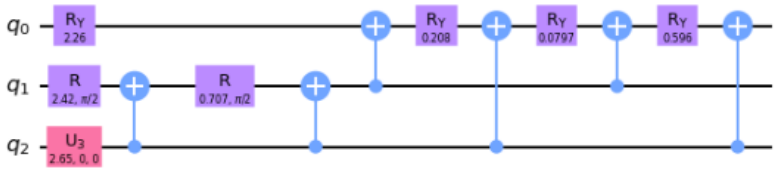
Se han usado tres distribuciones de probabilidad, cada tiene una correspondiente implementación en Qiskit que la carga, en concreto en el módulo de *circuit.library* (enlace). Tanto la distribución Normal como la distribución normal logarítmica tienen media $\mu = 0$ y desviación estándar $\sigma = 0.15$. En la figura 5.1 se observa los circuitos usados para cargarlas.

5.2.2. Espacio Latente y Discriminador

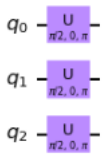
Estos dos elementos son comunes a todas las arquitecturas. El espacio latente básicamente es un generador de números pseudoaleatorios en el rango de $[-\pi, \pi]$. La estructura de puertas del discriminador se puede apreciar en la figura 5.2, la cual consta de puertas Hadamard y rotaciones en los ejes X, Y, Z aplicadas a todos los qubits. A continuación, se aplican dos puertas CX, la primera entre los qubits 1 y 2, y la segunda del qubit 2 al 3. Finalmente, se aplican nuevamente rotaciones R_X, R_Y, R_Z al último qubit, que es el de salida.



(a) Estructura de puertas que carga una distribución Normal



(b) Estructura de puertas que carga una distribución Normal Logarítmica



(c) Estructura de puertas que carga una distribución Uniforme

Figura 5.1: Estructuras de puertas que cargan las distribuciones de datos reales

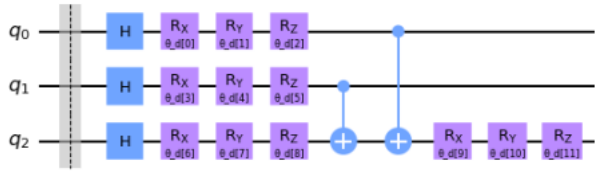


Figura 5.2: Esquema de la estructura de puertas del discriminador.

5.2.3. Generador y Arquitecturas

El generador, como se mencionó anteriormente, es responsable de generar datos sintéticos. En la experimentación se han usado tres tipos de generadores diferentes, conformando tres arquitecturas distintas y se convierte en el elemento diferenciador entre ellas. En lo que sigue se nombrarán a los modelos como *TwoLocal*, *ES* y *Gendis*. Es importante tener en cuenta que los dos primeros son circuito parametrizados implementados en *Qiskit*, por lo que para evitar confusión se indicará explícitamente si se hace referencia a ellos como los modelos o como los circuitos parametrizados.

Modelo I: TwoLocal

La arquitectura usada para el generador en este primer modelo se puede visualizar en la figura 5.3 y corresponde a la implementación del circuito *TwoLocal* de Qiskit. Este circuito consta de dos capas y puertas CZ para el entrelazamiento. Las puertas parametrizadas usadas son rotaciones R_Y y R_Z. En general está implementación viene inspirada de Quiskit (enlace).

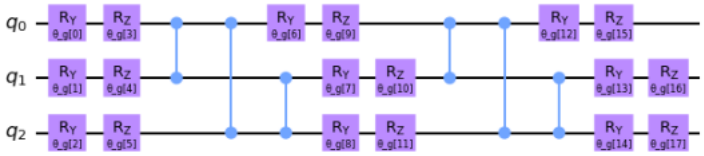


Figura 5.3: Estructura de puertas del generador en el primer modelo usando una instancia de *TwoLocal*.

Modelo II: ES

Esta arquitectura usa el circuito parametrizado *EfficientSU2* implementado en *Qiskit*, que, al igual que el anterior se encuentra en el módulo *Qiskit.library*

(enlace). Se ha probado tres veces, variando el número de capas, $L = 1$, $L = 2$ y $L = 6$, como se muestra en las figuras 5.4a, 5.4b y 5.4c respectivamente. La arquitectura general se compone de dos puertas R_Y y R_Z en cada uno de los qubtis. Y a continuación, por cada capa, le siguen puertas CX, en particular una que conecta los qubits q_0 (*control*) con q_1 (*controlado*) y otra que conecta los qubits q_1 (*control)* con q_2 (*controlado*) seguidas nuevamente de rotaciones R_Y y R_Z en cada qubit.

Modelo III: Gendi

En esta arquitectura el generador presenta el mismo esquema que el discriminador (Fig. 5.2).El motivo por el que se ha usado es estudiar el comportamiento del sistema bajo una arquitectura de puertas.

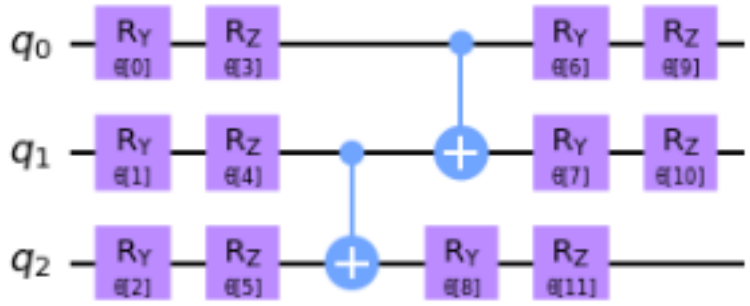
5.3. Metodología

Esta sección discute la metodología seguida de cara a la implementación. En el subapartado 5.3.1 se detalla cómo se han enlazado los circuitos para poder llevar a cabo la tarea de entrenamiento. En el subapartado 5.3.2 se explica la función de coste empleadas tanto en el discriminador como en el generador así como la métrica usada para evaluar el rendimiento de los modelos.

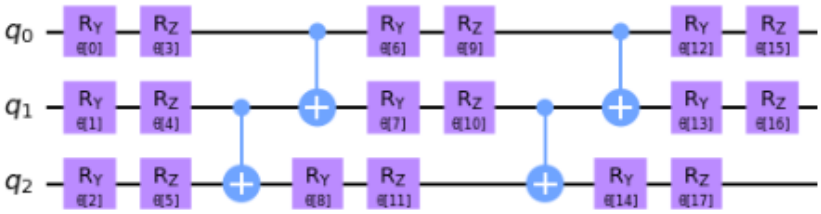
5.3.1. Compilación de circuitos

Como se indica en la descripción de la figura 4.1, es necesario usar dos circuitos debido a la naturaleza cuántica de los mismos. Por una parte, se tiene un circuito que concatena el circuito que carga la distribución real de los datos con el discriminador, *real_disc_circuit* Fig.(5.5a) , y por otra parte, otro que concatena el circuito del generador con el del discriminador, *gen_disc_circuit* Fig.(5.5b).

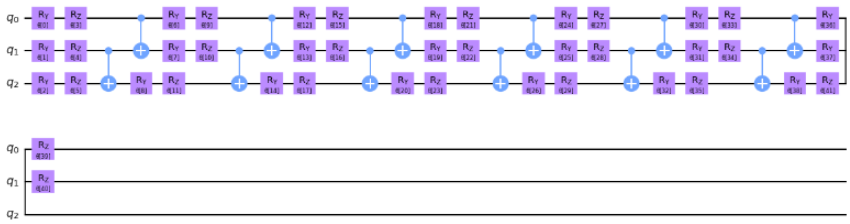
Tras la creación de estos dos circuitos se crean tres instancias de la clase *CircuitQNN*, perteneciente al módulo *qiskit_machine_learning.neural_network* de *Qiskit*. Esto es necesario ya que ayuda bastante con los cálculos del gradiente. Por otra parte, permite congelar los pesos de los circuitos, siendo necesario ya que el circuito *gen_disc_circuit* tiene tanto los parámetros del generador, como los del discriminador, y durante el entrenamiento de uno es necesario que los pesos del otro estén congelados. También hay que tener en cuenta que el discriminador recibe dos tipos de entradas, una con los datos reales por



(a) Estructura de puertas del generador en el segundo modelo usando una capa.

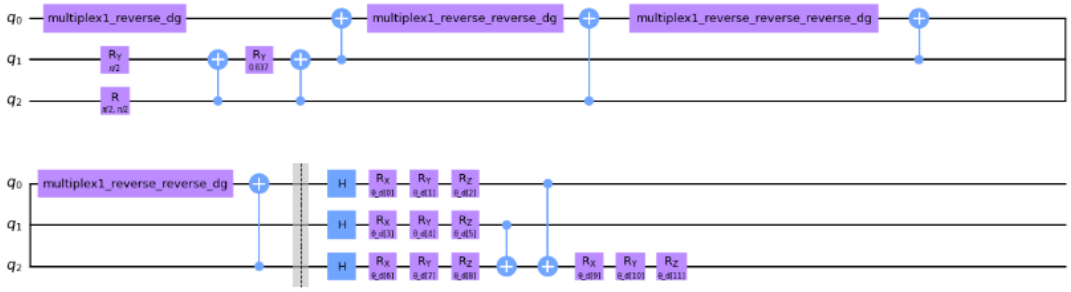


(b) Estructura de puertas del generador en el segundo modelo usando dos capa.

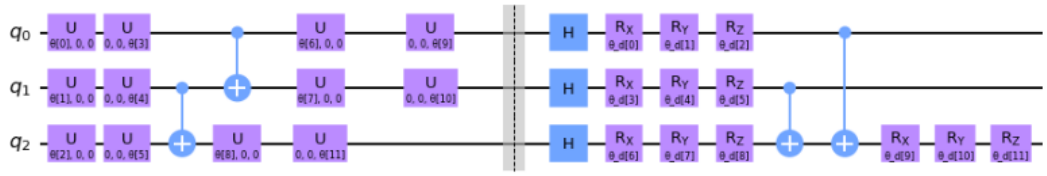


(c) Estructura de puertas del generador en el segundo modelo usando seis capa.

Figura 5.4: Estructuras del generador usadas en el segundo modelo. Todas ellas son instancias de EfficientSU2 pero variando el número de capas.



(a) (*real_disc_circuit*)Concatenación del circuito que carga los datos de una distribución, en este caso la distribución normal, con el del discriminador



(b) (gen_disc_dicruit)Concatenación del circuito del generador con el del discriminador

Figura 5.5: Circuitos concatenados

parte de la distribución de datos, y otra con los datos sintéticos del generador.

Entonces se definen tres instancias de *CircuitQNN*. Una llamada *disc_fake_qnn* que parte de *gen_disc_circuit*, en donde los pesos del generador están congelados y solo se actualizarán los del discriminador. Otra nombrada *disc_real_qnn*, que parte del circuito *real_disc_circuit*, ya que en este caso la entrada al discriminador serán los datos reales. Por último, se define una tercera instancia llamada *gen_qnn*, que parte de *gen_disc_circuit*, y en donde los pesos del discriminador están congelados para que los del generador puedan actualizarse.

Por último comentar que la dinámica del entrenamiento no difiere de la parte clásica. En cada paso de entrenamiento se entrena el discriminador a razón 5 : 1 con respecto el generador.

5.3.2. Métrica y función de pérdida

Aunque la función de perdida que viene dada por la ecuación (4.1) sea una única función, el problema de minimización requiere minimizar tanto el discri-

minador como el generador, y de manera adversaria. Entonces ha de definirse una función de coste para cada uno. La función de coste para el generador es

$$L^G = -Pr\Big(D(\theta_D, G(\theta_G)) = |real\rangle\Big)$$

mientras que para el discriminador es,

$$L^D = Pr\Big(D(\theta_D, G(\theta_G)) = |real\rangle\Big) - Pr\Big(D(\theta_d, R) = |real\rangle\Big)$$

que al combinarlas resultan en la ecuación (4.1).

En la implementación se ha aprovechado la clase *Statevector*, que viene dada en el módulo *qiskit.quantum_info*, la cual tiene un método que nos da las probabilidades, *probabilities()*, facilitando los cálculos.

La métrica usada para la comparación de modelos ha sido la divergencia de Kullback-Leibler [KL51]. Ésta es una medida no simétrica que indica la similitud o diferencia existente entre dos distribuciones de probabilidad, donde una se considera una aproximación de la otra. Destacar que aunque a menudo se considera una métrica o distancia, en realidad no lo es ya que no cumple la propiedad simétrica.

Definición 5.1 (Divergencia KL)**.** Sean P y Q dos distribuciones de probabilidad de una variable aleatoria discreta que tal que $Q(i) > 0$ para cualquier i que verifique $P(i) > 0$, su divergencia KL se define como

$$D_{KL}(P;Q) = \sum_i P(i) \ln\left(\frac{P(i)}{Q(i)}\right)$$

en caso de que las distribuciones sean de una variable aleatoria continua, la divergencia KL se define como la siguiente integral

$$D_{KL}(P;Q) = \int_{-\infty}^{\infty} p(x) \ln\left(\frac{p(x)}{q(x)}\right)$$

siendo p y q las respectivas funciones de densidad de las distribuciones Q y P.

La divergencia KL va a ser usada en un contexto referente a distribuciones de probabilidad de dos conjuntos de bits. Más concretamente, dado dos conjuntos de bits A y B, la divergencia KL entre ellos mide la cantidad de

información adicional necesaria para representa la distribución de probabilidad del conjunto A usando la distribución del conjunto B. En general, la divergencia proporciona una medida de cuánto difiere el conjunto A del conjunto B, en términos de sus probabilidades de ocurrencia para cada posible combinación de bits.

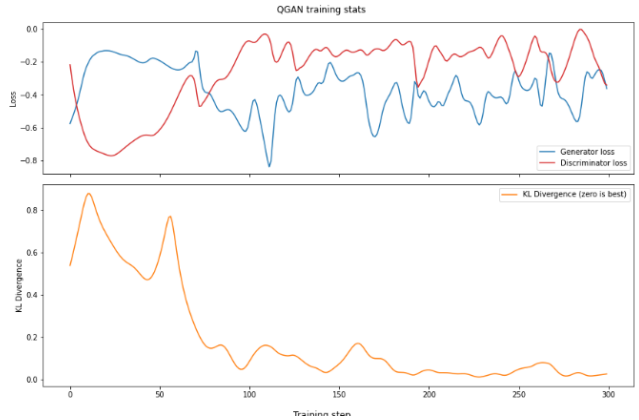
5.4. Resultados y Análisis

En esta sección, se presentarán los resultados de los experimentos en el apartado 5.4.1 mediante tablas, gráficas que muestran tanto las curvas de las funciones de coste del generador y el discriminador en el entrenamiento como de la métrica. También se incluyen histogramas que recogen la distribución de datos sintética que genera el generador usando los mejores pesos durante el entrenamiento. En el apartado 5.4.2 se procederá a analizar dichos resultados y a realizar una comparativa entre las distintas arquitecturas.

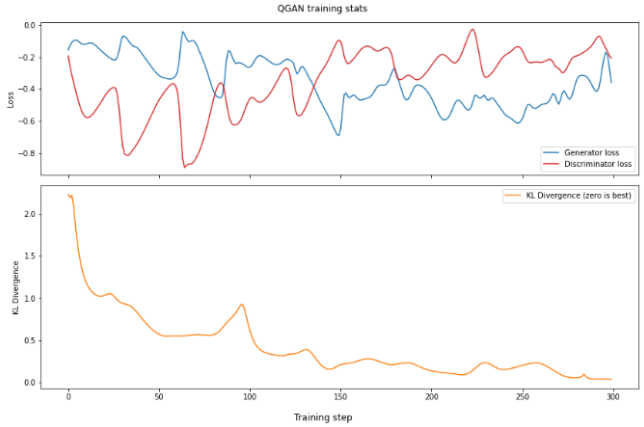
5.4.1. Resultados

		TwoLocal	ES1	ES2	ES3	Gendi
Normal	Gen. Cost	-0,494	-2,28E-22	-4,56E-12	-0,043	-2,63E-05
	Discr. Cost	-0,27	-0,35	-0,474	-0,423	-0,597
	KL Div	0,107	0,594	0,643	0,838	3,124
LogNormal	Gen. Cost	-0,538	-6,62E-31	-7,58E-26	-0,006	-8,51E-25
	Discr. Cost	-0,099	-0,533	-0,487	-0,469	-0,647
	KL Div	0,158	0,333	2,229	1,864	0,772
Uniforme	Gen. Cost	-0,268	-1,37E-30	-1,14E-15	-0,026	-9,91E-29
	Discr. Cost	-0,313	-0,379	-0,476	-0,583	-0,662
	KL Div	0,052	0,142	1,085	0,553	2,259

Tabla 5.1: Resultados obtenidos al final del entrenamiento



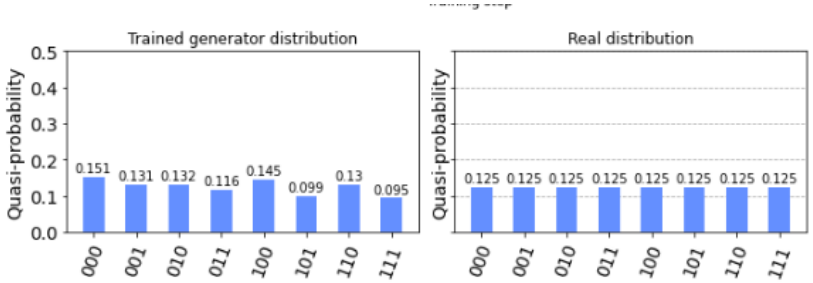
(a) Distribución Uniforme



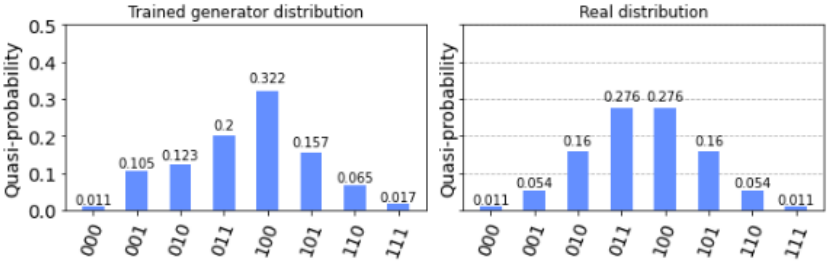
(b) Distribución Normal

(c) Distribución LogNormal

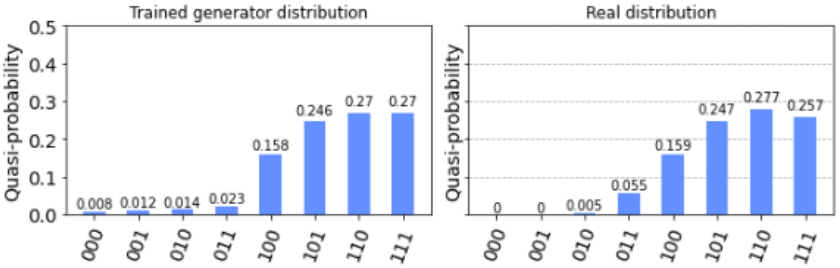
Figura 5.6: (Modelo *TwoLocal*): Los paneles de la parte superior representan las curvas de entrenamiento y los de la parte inferior la evolución de la métrica durante el proceso de entrenamiento.



(a) Distribución Uniforme

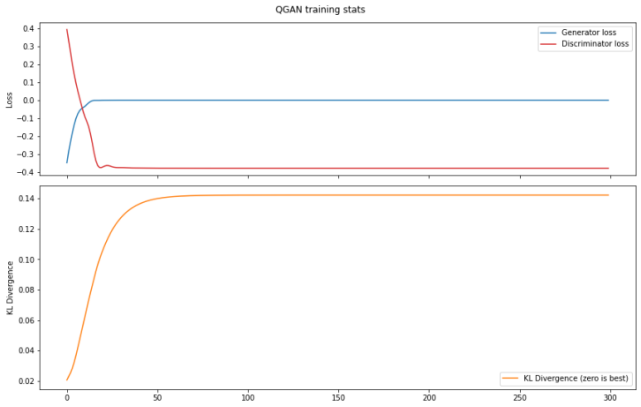


(b) Distribución Normal

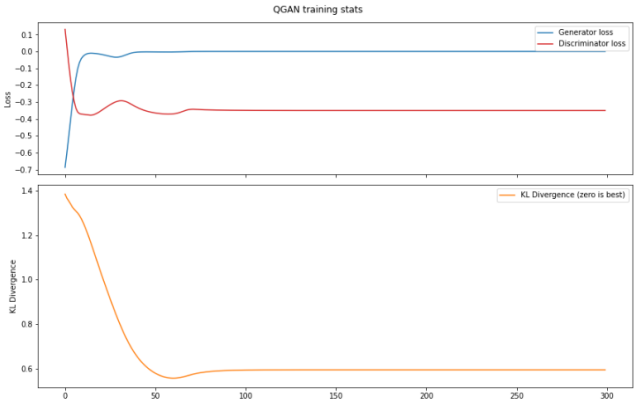


(c) Distribución LogNormal

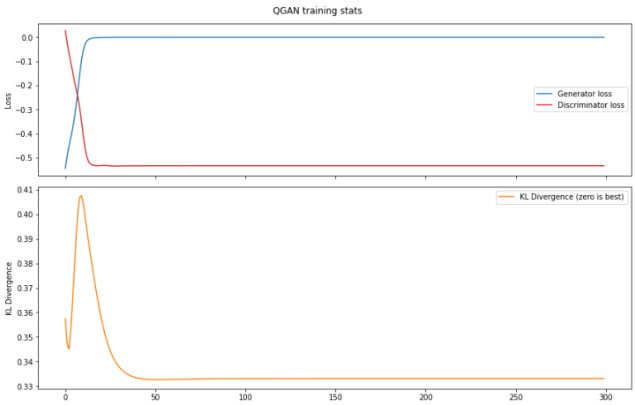
Figura 5.7: (Modelo *TwoLocal*): Histogramas en donde los paneles de la parte izquierda representan la distribución de los datos que el generador muestrea y los de la parte derecha la distribución real de datos.



(a) Distribución Uniforme

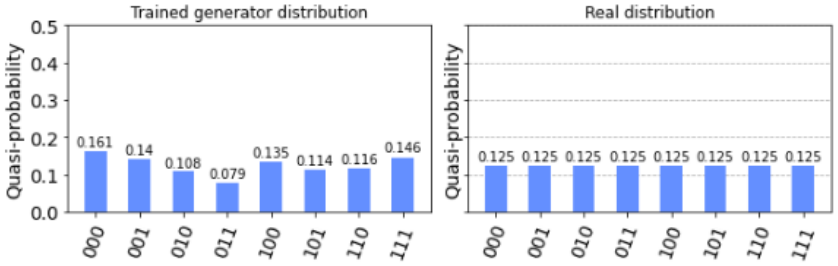


(b) Distribución Normal

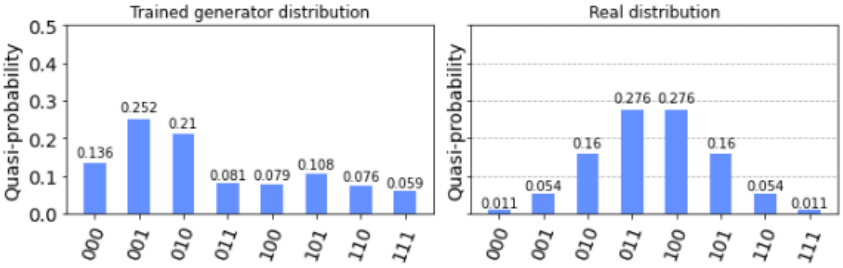


(c) Distribución LogNormal

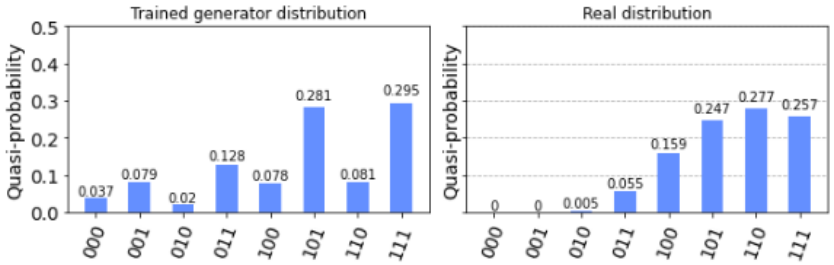
Figura 5.8: (Modelo *EffientSU2* con $L = 1$): Los paneles de la parte superior representan las curvas de entrenamiento y los de la parte inferior la evolución de la métrica durante el proceso de entrenamiento.



(a) Distribución Uniforme

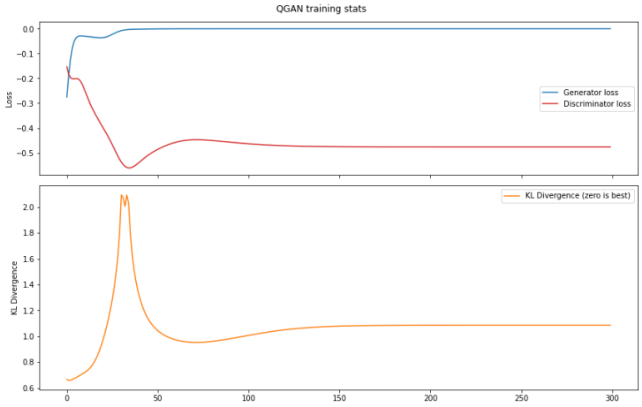


(b) Distribución Normal

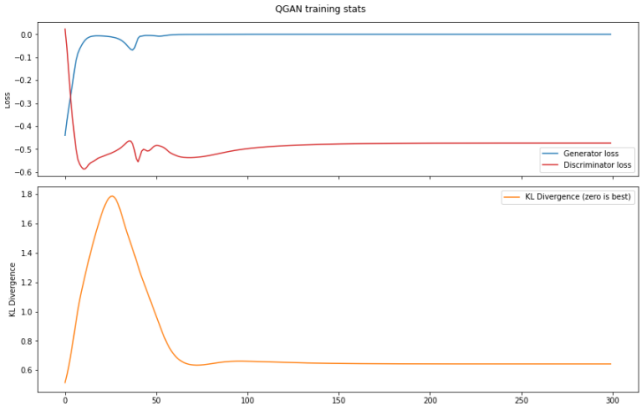


(c) Distribución LogNormal

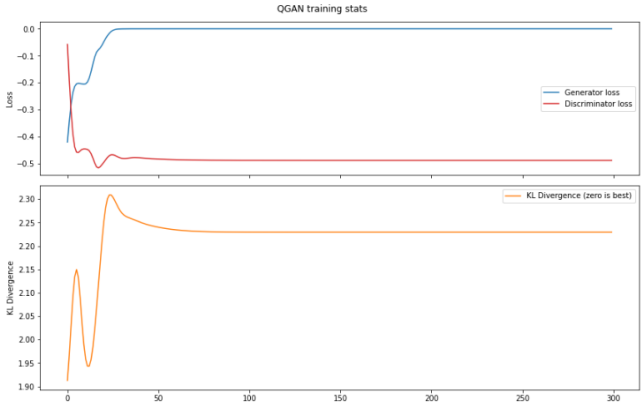
Figura 5.9: (Modelo *EfficientSU2 con* $L = 1$): Histogramas en donde los paneles de la parte izquierda representan la distribución de los datos que el generador muestrea y los de la parte derecha la distribución real de datos.



(a) Distribución Uniforme

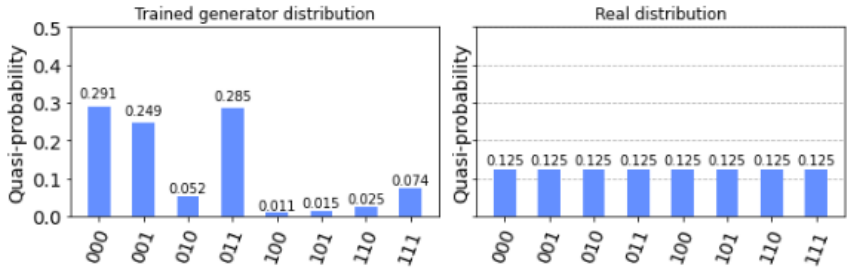


(b) Distribución Normal

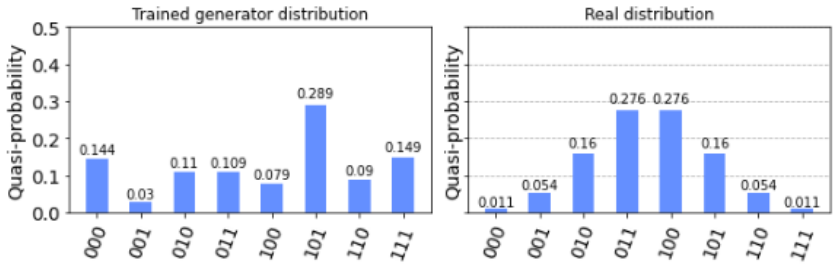


(c) Distribución LogNormal

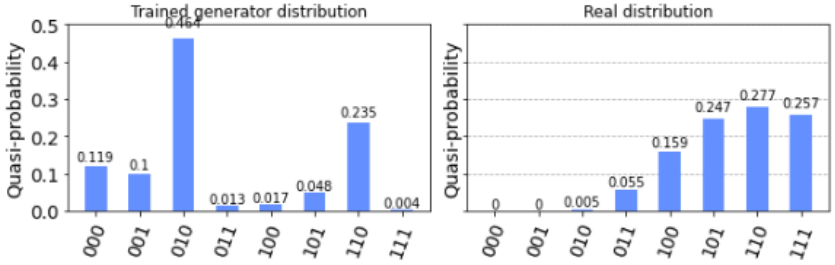
Figura 5.10: (Modelo *EfficientSU2 con* $L = 2$): Los paneles de la parte superior representan las curvas de entrenamiento y los de la parte inferior la evolución de la métrica durante el proceso de entrenamiento.



(a) Distribución Uniforme

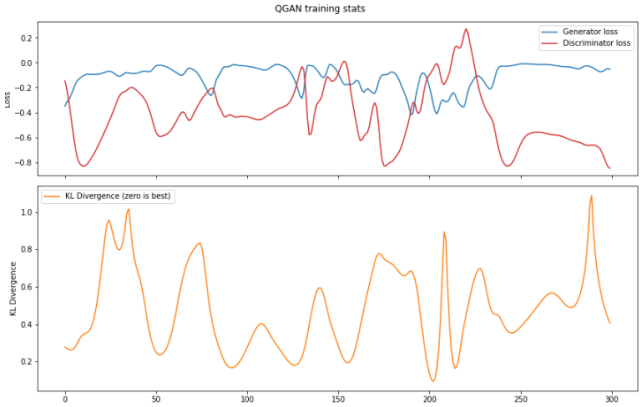


(b) Distribución Normal



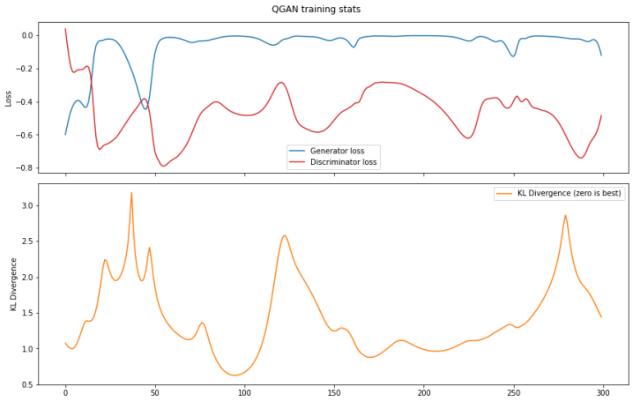
(c) Distribución LogNormal

Figura 5.11: (Modelo *EfficientSU2 con* $L = 2$): Histogramas en donde los paneles de la parte izquierda representan la distribución de los datos que el generador muestrea y los de la parte derecha la distribución real de datos.



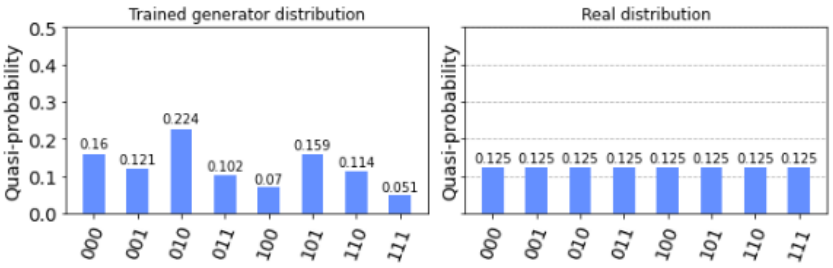
(a) Distribución Uniforme

(b) Distribución Normal

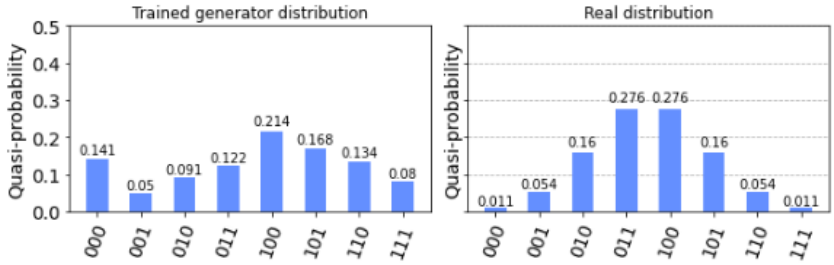


(c) Distribución LogNormal

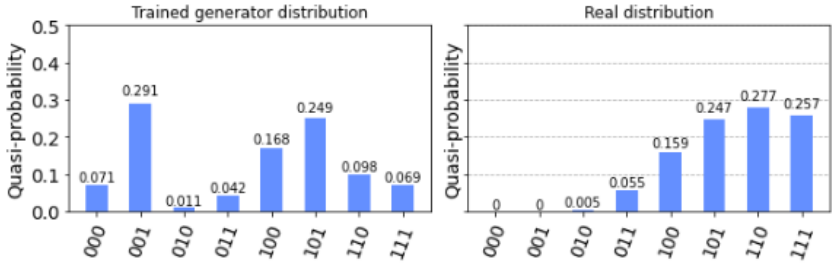
Figura 5.12: (Modelo *EfficientSU2 con* $L = 6$): Los paneles de la parte superior representan las curvas de entrenamiento y los de la parte inferior la evolución de la métrica durante el proceso de entrenamiento.



(a) Distribución Uniforme

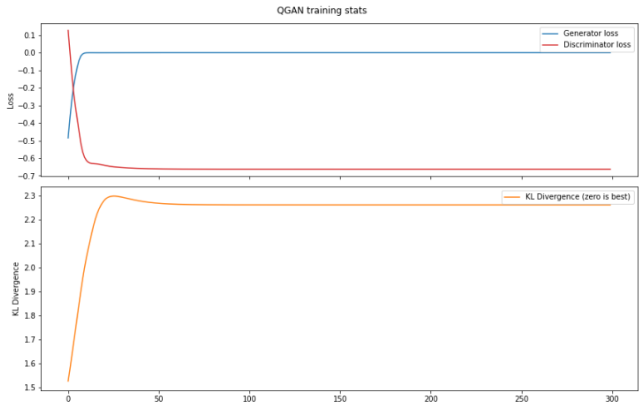


(b) Distribución Normal

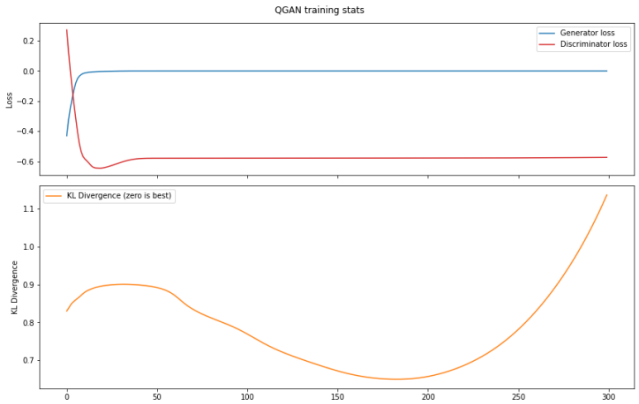


(c) Distribución LogNormal

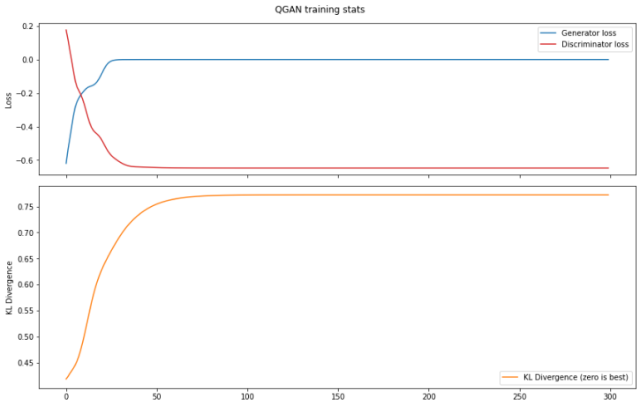
Figura 5.13: (Modelo *EfficientSU2 con* $L = 6$): Histogramas en donde los paneles de la parte izquierda representan la distribución de los datos que el generador muestrea y los de la parte derecha la distribución real de datos.



(a) Distribución Uniforme

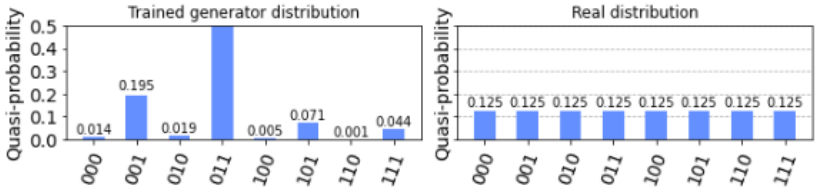


(b) Distribución Normal

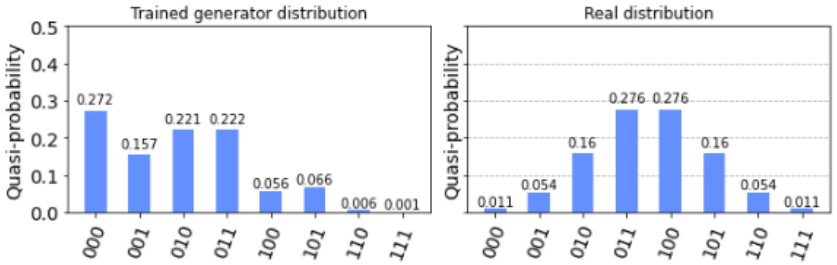


(c) Distribución LogNormal

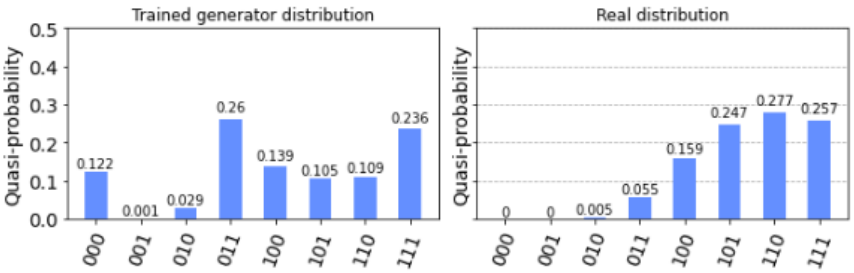
Figura 5.14: (Modelo *Gendi*): Los paneles de la parte superior representan las curvas de entrenamiento y los de la parte inferior la evolución de la métrica durante el proceso de entrenamiento.



(a) Distribución Uniforme



(b) Distribución Normal



(c) Distribución LogNormal

Figura 5.15: (Modelo *Gendi*): Histogramas en donde los paneles de la parte izquierda representan la distribución de los datos que el generador muestrea y los de la parte derecha la distribución real de datos.

5.4.2. Análisis

Como ya se comentó, se han usado tres arquitecturas *TwoLocal*, *ES* y *Gendi* para aprender tres distribuciones de probabilidad bien conocidas, distribución normal, normal logarítmica y uniforme. Antes de pasar con el análisis, conviene enfatizar de nuevo que los histogramas presentados más arribas están hechos con los mejores pesos obtenidos durante el entrenamiento, no son los pesos finales, dado que un entrenamiento en exceso puede resultar en un sobreajuste y empeorar los resultados.

En lo que respecta al modelo *TwoLocal*, se observa en 5.6b y en 5.6a que las curvas de la divergencia KL durante el entrenamiento tienen una tendencia descendente, lo que significa que a medida que avanza el entrenamiento el generador es capaz de reproducir mejor la distribución objetivo, que en este caso es la uniforme. En cuanto a la distribución normal logarítmica (Fig. 5.6c), se observa que la tendencia también es decreciente pero con fluctuaciones más acentuadas que presentan picos y valles más notables. En lo referente a las curvas de entrenamiento, puede apreciarse que en cierto modo son las tres muy similares, todas ellas presentan fluctuaciones con picos y valles bien acentuados sin subidas o bajadas ligeras. Idealmente, en las curvas de entrenamiento, el generador empieza con un error alto ya que no es capaz de producir buenas muestras y, poco a poco el error toma una tendencia hacia el valor cero, hasta que converge y se mantiene constante en un valor fijo, indicando que cada vez se aproxima mejor. En cambio, en las gráficas se observa que, para la distribución uniforme y para la normal logarítmica el generador consigue mejorar en cuanto al error pero no demasiado, y en la distribución normal empeora. No obstante, hay que tener en cuenta que las curvas se mueven en un entorno cercano al cero por lo que el error tampoco es muy grande. Finalmente en 5.7, se puede observar que al usar los mejores pesos durante el entrenamiento, las muestras generadas por el generador se parecen a las obtenidas de la distribución real, que aunque sean muy similares, no son idénticas. En resumen, esta arquitectura ha conseguido generar muestras sintéticas muy cercanas a la distribución objetivo por lo que ha sido todo un éxito.

El modelo *ES* se ha ejecutado tres veces variando el hiperparámero del número de capas. Se tiene una ejecución con una, dos y seis capas en la arquitectura del generador, conformando tres versiones distintas. Al no tener ninguna referencia como el modelo anterior, se han usado varias capas para probar cual es la óptima.

En cuanto a la ejecución con una capa *ES1*, se observa en las gráficas de la figura 5.8 que se produce un estancamiento en el proceso de entrenamiento, es decir, converge tempranamente. Esto puede deberse a varios factores, entre ellos una posibilidad son las *barren plateaus* mencionadas anteriormente, ya que al encontrarse los gradientes en un entorno del cero en casi todo punto del espacio paramétrico, no hay mucho margen de movimiento en lo que respecta a estas curvas. El discriminador no ha podido entrenarse adecuadamente, haciendo que el generador no reciba buena retroalimentación y no sea capaz de generar datos cercanos a los reales. En las curvas de la divergencia

KL, se puede apreciar una tendencia decreciente en lo que respecta a las distribuciones normal y normal logarítmica, por lo que pese al estancamiento en el entrenamiento, el modelo ha sido capaz de aproximarse un poco a ellas. De hecho, en los histogramas (Fig. 5.9) se observa que, ni usando los mejores pesos se ha podido reproducir una distribución cercana a la objetivo. No obstante, la distribución generada por el generador tiene una tendencia parecida a la objetivo, pero dista mucho de ella. En resumen, esta arquitectura ha sido capaz de aproximarse a las distribuciones de datos reales pero distan de estar cercanas, por lo que los resultados obtenidos no son lo suficientemente fructíferos.

Usando dos capas, *ES2*, se observa algo parecido al caso anterior en las curvas de entrenamiento (Fig. 5.10), donde se aprecia una convergencia prematura. De hecho, la separación entre la curva del discriminador y del generador es mucho más notable y el discriminador presenta un mayor error. En lo que respecta a la curva de la divergencia KL, se observa una tendencia ascendente, por lo que la distribución generada se aleja cada vez más de la objetivo. El modelo no aprende nada durante el entrenamiento, de hecho su conocimiento empeora durante el mismo. En los histogramas (Fig. 5.11) se observa esto claramente, las distribuciones generadas y las objetivo no se parecen en absoluto, al contrario que el caso anterior. En resumen, en este caso los resultados no pésimos.

En cuanto a la ejecución usando seis capas, *ES6*, notar que las curvas de entrenamiento (Fig. 5.12) son totalmente distintas con respecto a los modelos *ES1* y *ES2*. En este caso las curvas ya no convergen prematuramente, por lo que el proceso de entrenamiento no sufre un estancamiento. Las curvas de la divergencia KL ya no son tan monótonas como en los casos anteriores, por lo que la tendencia no está muy clara. Las fluctuaciones son muy pronunciadas. El mínimo absoluto de esas curvas indica que en ese momento, la distancia entre la distribución objetivo y generada es la mínima del todo el entrenamiento y, a partir de ahí ésta se amplía dando peores resultados. Tomando los mejores pesos obtenidos en el entrenamiento, se observa en los histogramas (Fig. 5.13) que las distribuciones generadas por el generador se asemejan un poco a las de las distribuciones reales. No obstante y pese a que haya cierta semejanza, la aproximación con respecto a la distribución normal logarítmica (Fig. 5.13c) y a la uniforme (Fig. 5.13a) tampoco son buenas. En cambio, la distribución normal (Fig. 5.13b) si es más similar. En resumen, estos resultados no son tan malos como los anteriores, producen aproximaciones semejantes, aunque no lo suficientemente buenos en algunos casos.

Por último, el modelo *Gendi* presenta unas curvas en los errores del discriminador y del generador (Fig. 5.14) similares a los modelos *ES1* y *ES2*, con una convergencia prematura. Las curvas de la divergencia KL en los casos de la distribución uniforme (Fig. 5.14a) y normal logarítmica (Fig. 5.14c) son monótonas crecientes, indicando que lo que hace el entrenamiento es empeorar la aproximación de ambas distribuciones en todo momento. De hecho, al inicio del mismo se encuentran los mínimos absolutos. En cuanto a la distribución normal, se observa un comportamiento parabólico en la curva de la divergencia KL, el entrenamiento llega al mínimo absoluto en torno a la iteración *170*, siendo la mejor aproximación,y explotando a partir de ahí. Tomando los mejores pesos las distribuciones generadas, se observa en los histogramas (Fig. 5.15) que las distribuciones generadas y las objetivo no se parecen en nada. En resumen, esta aproximación tampoco es buena.

En vistas de la tablea 5.1, se puede observar que entre todos los modelos, el que mejor métrica presenta y, por tanto, el que mejor consigue aproximar todas las distribuciones generadas a las reales, es el modelo *TWolocal*. El que peor aproximaciones presenta de cara a la distribución normal y uniforme es el modelo *Gendi*, y el que peor lo hace con respecto a la distribución normal logarítmica es el modelo *ES2*. Estos resultados son los obtenidos a finales el entrenamiento, pero si se toman en cuenta los pesos óptimo obtenidos durante el dicho proceso, cabría destacar que el modelo *TwoLocal* seguiría siendo el mejor y a continuación iría el modelo *ES6*, pues observando los histogramas (Fig. 5.13) es el segundo que mejor se aproxima.

5.5. Conclusiones

La finalidad de este proyecto es exponer teóricamente el funcionamiento de una QGAN. Pero previamente, y para introducir al lector en esta rama de la inteligencia artificial y la computación cuántica, unos capítulos previos explicando el funcionamiento de las GANs clásicas, los circuitos variacionales y la codificación de datos clásicos en sistemas cuánticos han sido necesarios.

En cuanto el caso práctico, se ha tenido en cuenta que las QGANs aún están en sus primeras etapas de adopción y aplicación en diversas áreas. Se ha optado por unos pequeños experimentos en los que se han usado tres arquitecturas, y en los que se pretendían generar datos sintéticos cuyas distri-

buciones se aproximen a otras distribuciones de probabilidad bien conocidas. En los experimentos se observa que, las aproximaciones no son muy buenas, a excepción de una de ellas. Por lo que se intuye que aún se necesita más avances y estudios en esta área de cara a poder llevar a las QGANs a aplicaciones reales, donde las distribuciones de probabilidades que se desean aproximar son mucho más complejas y desconocidas.

5.6. Trabajos futuros

Como trabajos futuros, se propone continuar en el estudio de las QGANs. Mejorar los métodos de codificación de datos en los circuitos variacioanales, dado que es un aspecto crítico, y realizar varios estudios probándolos en QGANs. También se propone aplicarlas a problemas de imágenes usando una codificación de datos eficiente y consistente con el objeto de estudiar su capacidad al incrementar la dificultad del problema. Por último, sería interesante hacer varias comparatiavs de QGANs con sus contrapartes clásicas, usando varias métricas, en busca de alguna ventaja por parte de la versión cuántica.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[AB18] Karen Simonyan Andrew Brock, Jeff Donahue. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. [Citado en pág. 1]

[AL98] Daniel S Abrams and Seth Lloyd. Nonlinear quantum mechanics implies polynomial-time solution for np-complete and# p problems. *Physical Review Letters*, 81(18):3992, 1998. [Citado en pág. 27]

[bai] baidu. qmlbaidu. [Citado en pág. 29]

[Bal12] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012. [Citado en pág. 1]

[BLSF19] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019. [Citado en pág. 16]

[BMN+21] Ryan Babbush, Jarrod R McClean, Michael Newman, Craig Gidney, Sergio Boixo, and Hartmut Neven. Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX Quantum*, 2(1):010103, 2021. [Citado en pág. 36]

[CGAG17] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Quantum neuron: an elementary building block for machine learning on quantum computers. 2017. [Citado en pág. 30]

[Cyb89] George Cybenko. *Approximation by superpositions of a sigmoidal function*. Springer, first edition, 1989. [Citado en págs. 18 and 19]

[CYQ+20] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. *IEEE Access*, 8:141007–141024, 2020. [Citado en pág. 17]

[DBK+22] Andrew J Daley, Immanuel Bloch, Christian Kokail, Stuart Flannigan, Natalie Pearson, Matthias Troyer, and Peter Zoller. Practical quantum advantage in quantum simulation. *Nature*, 607(7920):667–676, 2022. [Citado en pág. 13]

[Doe16] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016. [Citado en págs. 1 and 2]

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014. [Citado en págs. 1, 2, and 35]

[HCSS21] Zoë Holmes, Nolan Coble, Andrew T. Sornborger, and Yiğit Subaşı. On nonlinear transformations in quantum computation. 2021. [Citado en pág. 19]

[HCT+19] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019. [Citado en pág. 32]

[Hin07] Geoffrey E Hinton. Boltzmann machine. *Scholarpedia*, 2(5):1668, 2007. [Citado en pág. 1]

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. [Citado en pág. 2]

[HM17] Aram W Harrow and Ashley Montanaro. Quantum computational supremacy. *Nature*, 549(7671):203–209, 2017. [Citado en pág. 13]

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006. [Citado en pág. 1]

[JKA21] Nisheeth Joshi, Pragya Katyayan, and Syed Afroz Ahmed. Comparing classical ml models with quantum ml models with parametrized circuits for sentiment analysis task. 1854(1):012032, 2021. [Citado en pág. 47]

[JRO+17] Peter D. Johnson, Jonathan Romero, Jonathan Olson, Yudong Cao, and Alán Aspuru-Guzik. Qvector: an algorithm for device-tailored quantum error correction. 2017. [Citado en pág. 36]

[KKP21] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. Quantum computing methods for supervised learning. *Quantum Machine Intelligence*, 3(2):23, 2021. [Citado en pág. 35]

[KL51] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951. [Citado en pág. 54]

[LBR17] Austin P Lund, Michael J Bremner, and Timothy C Ralph. Quantum sampling problems, bosonsampling and quantum supremacy. *npj Quantum Information*, 3(1):15, 2017. [Citado en pág. 13]

[LC20] Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Physical Review A*, 102(3), September 2020. [Citado en págs. 26, 30, and 31]

[LTR17] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168:1223–1247, 2017. [Citado en pág. 18]

[LWX+20] Yong Liu, Dongyang Wang, Shichuan Xue, Anqi Huang, Xiang Fu, Xiaogang Qiang, Ping Xu, He-Liang Huang, Mingtang Deng, Chu Guo, et al. Variational quantum circuits for quantum state tomography. *Physical Review A*, 101(5):052316, 2020. [Citado en pág. 17]

[MBS+18] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), November 2018. [Citado en págs. 20 and 21]

[McK22] McKinsey. Mckinsey digital. 2022. [Citado en pág. 35]

[MSJ+15] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015. [Citado en pág. 1]

[MVBS04] Mikko Mottonen, Juha J Vartiainen, Ville Bergholm, and Martti M Salomaa. Transformation of quantum states using uniformly controlled rotations. *arXiv preprint quant-ph/0407010*, 2004. [Citado en pág. 28]

[Pea63] E. S. Pearson. Some problems arising in approximating to probability distributions, using moments. *Biometrika*, 50(1/2):95, June 1963. [Citado en pág. 2]

[Pen] PennyLane. Pennylane official web site. [Citado en pág. 29]

[Per89] Asher Peres. Nonlinear variants of schrödinger's equation violate the second law of thermodynamics. *Physical Review Letters*, 63(10):1114, 1989. [Citado en pág. 27]

[PMS+14] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), July 2014. [Citado en pág. 36]

[Qui20a] Quiskit Quiskit. Quiskit documentation. *Quiskit Machine Learning Documentation*, 2020. [Citado en pág. 32]

[Qui20b] Quiskit Quiskit. Quiskit documentation. *Quiskit Machine Learning Documentation*, 2020. [Citado en pág. 32]

[RM15] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015. [Citado en pág. 2]

[ROAG17] Jonathan Romero, Jonathan P Olson, and Alan Aspuru-Guzik. Quantum autoencoders for efficient compression of quantum data. *Quantum Science and Technology*, 2(4):045001, August 2017. [Citado en págs. 32 and 36]

[SDWMG15] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015. [Citado en pág. 2]

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999. [Citado en pág. 36]

[SMH07] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798, 2007. [Citado en pág. 1]

[TK19] Timo Aila Tero Karras, Samuli Laine. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2019. [Citado en pág. 1]

[Tru01] Carlo A Trugenberger. Probabilistic quantum memories. *Physical Review Letters*, 87(6):067901, 2001. [Citado en pág. 26]

[TT13] Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013. [Citado en pág. 2]

[UML13] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: The real-valued neural autoregressive density-estimator. *Advances in Neural Information Processing Systems*, 26, 2013. [Citado en pág. 2]

[VDOKK16] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016. [Citado en pág. 2]

[VM00] Dan Ventura and Tony Martinez. Quantum associative memory. *Information sciences*, 124(1-4):273–296, 2000. [Citado en pág. 26]

[ZLX+22] Shichao Zhang, Jiaye Li, Hang Xu, Hao Yu, and Jinjing Shi. Robust quantum feature selection algorithm. November 2022. [Citado en pág. 17]

[ZMS+23] Christa Zoufal, Ryan V Mishmash, Nitin Sharma, Niraj Kumar, Aashish Sheshadri, Amol Deshmukh, Noelle Ibrahim, Julien Gacon, and Stefan Woerner. Variational quantum algorithm for unconstrained black box binary optimization: Application to feature selection. *Quantum*, 7:909, 2023. [Citado en pág. 17]

[ZZM21] Pengzhan Zhao, Jianjun Zhao, and Lei Ma. Identifying bug patterns in quantum programs. pages 16–21, 2021. [Citado en pág. 47]