PRÁCTICA ACADÉMICA EN GRUPO BERNIER SAS: PERCEPTRÓN MULTICAPA EN COMPUTACIÓN CUÁNTICA

IVÁN DANIEL LOZANO JOYA

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍAS
INGENIERÍA MECATRÓNICA
PRÁCTICA ACADÉMICA

BUCARAMANGA – SANTANDER
SEPTIEMBRE 2020.

PRÁCTICA ACADÉMICA EN GRUPO BERNIER SAS: PERCEPTRÓN MULTICAPA EN COMPUTACIÓN CUÁNTICA

IVÁN DANIEL LOZANO JOYA

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍAS
INGENIERÍA MECATRÓNICA
PRÁCTICA ACADÉMICA

BUCARAMANGA – SANTANDER
SEPTIEMBRE 2020.

PRÁCTICA ACADÉMICA EN GRUPO BERNIER SAS: PERCEPTRÓN MULTICAPA EN COMPUTACIÓN CUÁNTICA

IVÁN DANIEL LOZANO JOYA

Informe final de practica académica para optar por el título de ingeniero en mecatrónica.

Directora

M.Eng. MSc. Jessica Gissella Maradey Lázaro

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA

FACULTAD DE INGENIERÍAS

INGENIERÍA MECATRÓNICA

PRÁCTICA ACADÉMICA

BUCARAMANGA – SANTANDER

SEPTIEMBRE 2020.

Septiembre de 2020

Por medio de este documento se autoriza la publicación de la información enunciada en este documento, aclarando que este fue sujeto a correcciones realizadas por la Ingeniera Jessica Gissella Maradey Lázaro, directora de la practica académica realizada en la empresa Grupo Bernier SAS por el estudianteIván Daniel Lozano Joya en el periodo 202010 para la obtención del título de ingeniero mecatrónico en la Universidad Autónoma de Bucaramanga UNAB.

En constancia firma:

Estudiante:	Directora:
Ivan Daniel Lozano Joya	Ing. Jessica Gissella Maradey

DEDICATORIA

Este libro lo dedico primeramente a Dios quien supo guiarme, darme las capacidades y la fuerza necesaria para completar mis metas y siempre levantarme antes los obstáculos que se presentaron en este camino de aprendizaje constante.

Agradezco a mi familia por el apoyo en todo este proceso, por la confianza que siempre ha depositado en mí, además del ejemplo y la enseñanza que me ha inculcado de siempre seguir adelante y buscar la mejora constante en todos los aspectos de la vida.

Adicionalmente, agradezco a la empresa Grupo Bernier SAS, por darme la oportunidad de mostrar mis capacidades y de darme una primera muestra del ambiente laboral y profesional, además, agradezco a mis profesores y a la universidad autónoma de Bucaramanga por suministrarme todos los conocimientos necesarios para desempeñar mi trabajo en mi práctica académica yen mi futuro profesional.

RESUMEN

Este artículo muestra una alternativa de perceptrón diseñada en computación cuántica, que consta de dos neuronas de entrada, dos neuronas en su capa oculta y una neurona en su capa de salida. El objetivo principal de este trabajo es el diseño de un clasificador de puntos ubicado dentro de un plano cartesiano, que depende de dos variables de entrada x, y; Estos se clasifican en dos tipos (tipo 1 ytipo 0). Este diseño se basa en el término distancia entre estados cuánticos, por lo que el programa determina si el punto en cuestión es de tipo 0 o tipo 1, dependiendo de la probabilidad de ser de un tipo u otro. La idea principal del trabajo gira en torno a la esfera cuántica, donde los valores 1 y 0 se asignan a los puntos estipulados por el usuario (Network Training). Luego, dependiendo de la distancia entre los puntos de entrenamiento y los puntos de prueba, las puertas cuánticas determinan si es más probable que sea de tipo cero o de tipo uno.

PALABRAS CLAVE:

Computación cuántica, red neuronal, perceptrón, Python, Qiskit, inteligencia artificial, esfera cuántica, estado cuántico, superposición cuántica.

ABSTRACT

This paper shows a perceptron alternative designed in quantum computing, which consists of two input neurons, two neurons in its hidden layer and a neuron in its output layer. The main objective of such work is the design of a point sorter located within a Cartesian plane, which depend on two input variables x, y; these are classified into two types (type 1 and type 0). This design is based on the term distance between quantum states, so the program determines whether the point in question is type 0 or type 1 depending on the probability of being of one type or another. The main idea of the work revolves around the quantum sphere, where values 1 and 0 are assigned to points stipulated by the user (Network Training). Then, depending on the distance between the training points and the test points, it is determined by quantum gates if it is more likely to be type zero or type one.

TABLA DE CONTENIDOS

1. INFORMACION GENERAL 1
1.1. INTRODUCCIÓN 1
1.2. OBJETIVOS 2
1.3. DESCRIPCIÓN DE LA ENTIDAD 3
1.3.1. INFORMACIÓN BÁSICA 3
1.3.2. SERVICIOS 3
1.3.3. CONTACTO CON EL JEFE INMEDIATO 5
2. DESARROLLO DE LA PRÁCTICA 6
2.1. MARCO CONCEPTUAL 6
2.1.1. INTELIGENCIA ARTIFICIAL 6
2.1.1.1. HISTORIA 7
2.1.1.2. PRINCIPIOS BÁSICOS 8
2.1.1.3. ¿PARA QUE SIRVE? 9
2.1.1.4. REDES NEURONALES 10
2.1.2. COMPUTACIÓN CUÁNTICA 18
2.1.3. PROGRAMACIÓN 25
3. ACTIVIDADES 36
4. METODOLOGÍA 37
5. RESULTADOS 39
6. CONCLUSIONES 60
7. OBSERVACIONES 60
8. BIBLIOGRAFÍA 61
9. APÉNDICE 64

LISTA DE TABLAS

Tabla 1. Cronograma de actividades manejado en la práctica.....36

Tabla 2. Información sobre avances en Deep Learning.....40

Tabla 3. Información sobre avances en Deep Learning parte 2.....43

Tabla 4. Tipos de algoritmos de Machine Learning.....44

Tabla 5. Algoritmos de perceptrón cuántico similares. (Bon, 2018;2019), (Schuld & Petruccione, 2014).....49

Tabla 6. Algoritmos de perceptrón cuántico similares 2. (Schuld, Fingerhunth, & Petruccion, Quantum Reseach Group, School of chemistry and physics, 2017) 50

Tabla 7. Mejoras y modificaciones en el algoritmo para implementar el perceptrón multicapa. (Lozano, 2020) 58

LISTA DE FIGURAS

Fig. 1. Estructura general de la red neuronal. (innovation, 2019).....11

Fig. 2. Clasificación con el perceptrón sencillo. (Bibing, s.f.).....15

Fig. 3. Definición del perceptrón, tomada textualmente. (Bibing, s.f.) 15

Fig. 4. Estructura del perceptrón multicapa. (Palacios, s.f.) 17

Fig. 5. Interfaz de Visual Studio. (Microsoft, 2019) 26

Fig. 6. Interfaz principal de Python.....28

Fig. 7. Inicialización de un qubit. (Microsoft, 2020) 30

Fig. 8. Instrucción for. (Microsoft, 2020) 31

Fig. 9. Código para importar librerías en Python. (Equipo de desarrollo de Qiskit , 2020) 32

Fig. 10. Inicialización del circuito cuántico. (Equipo de desarrollo de Qiskit , 2020) 33

Fig. 11. Aplicación de puertas cuánticas, Hadamard, CX y medición respectivamente. (Equipo de desarrollo de Qiskit , 2020).....33

Fig. 12. Visualización del circuito cuántico. (Equipo de desarrollo de Qiskit , 2020). 33

Fig. 13. Importación de complementos de simulación. (Equipo de desarrollo de Qiskit , 2020).....34

Fig. 14. Simulación del circuito cuántico. (Equipo de desarrollo de Qiskit , 2020). 34

Fig. 15. Histograma del resultado del circuito. (Equipo de desarrollo de Qiskit , 2020) 35

Fig. 16. Metodología en cascada. (Lozano, 2020) 37

Fig. 17. Diseño general del perceptrón simple. (Torres Barrán & Naveiro, 2019) 38

Fig. 18. Estructura general del perceptrón multicapa. (Burrueco, s.f.) 39

Fig. 19. Circuito cuántico 1.....50

Fig. 20. Circuito cuántico 2.....50

Fig. 21. Circuito cuántico implementado 51
Fig. 22. Importación de librerías. (Lozano, 2020).....51
Fig. 23. Inicialización de los vectores de entrenamiento. (Lozano, 2020) 52
Fig. 24. Inicialización del vector de entradas. (Lozano, 2020).....52
Fig. 25. Almacenamiento de los vectores de entrenamiento como un solo vector. (Lozano, 2020).....52
Fig. 26. Llamado de función general de clasificación. (Lozano, 2020).....52
Fig. 27. Función de inicialización de registros. (Lozano, 2020).....53
Fig. 28. Función de normalización de los vectores de entrada y entrenamiento. (Lozano, 2020).....53
Fig. 29. Función de simulación del circuito cuántico. (Lozano, 2020) 54
Fig. 30. Función de interpretación de los resultados obtenidos en la simulación. (Lozano, 2020).....55
Fig. 31. Finalización de la función de clasificación $classify()$. (Lozano, 2020).....55
Fig. 32. Finalización de la ejecución general del perceptrón y dibujo de los puntos clasificados en el plano x, y. (Lozano, 2020) 56
Fig. 33. Ejemplo de resultado final de clasificación mediante perceptrón cuántico sencillo. (Lozano, 2020).....57
Fig. 34. Ejemplo de la clasificación obtenida por el perceptrón multicapa. (Lozano, 2020) 59

1. INFORMACION GENERAL

1.1. INTRODUCCIÓN

Al pasar los años, los materiales y mecanismos para el diseño de máquinas de cálculo y procesamiento de información han venido cambiando, en la búsqueda de optimizar dichos procesos. Cada vez se obtiene mayor capacidad y velocidad, además de miniaturización de los materiales utilizados en los componentes de hardware con la cual se ha logrado que los circuitos electrónicos actuales sean de tamaño incluso nanométrico. Teniendo en cuenta que cada vez se busca mejorar los ítems mencionados, se ha optado por implementar la física cuántica en el acceso y programación de la información. Esta idea, aumenta en gran medida la velocidad y disminuye los errores que pueden verse reflejados en la computación clásica. A pesar de los grandes avances que se viene desarrollando todavía queda mucho camino por recorrer en cuanto a computación cuántica y a inteligencia artificial aplicando dicho principio, ya que actualmente se tienen muy pocas maquinas cuánticas pasa ejecutar los códigos desarrollados y los procesos que se han aplicado son los más elementales y sencillos.

En este trabajo se muestra el desarrollo de un perceptrón cuántico, organizado de tal forma que primero se realiza una primera muestra de los fundamentos y los elementos básicos de la inteligencia artificial en cuanto al perceptrón sencillo y al perceptrón multicapa, además de lo referente a computación cuántica para aplicar estos algoritmos.

Luego, se muestra un resumen de algunos avances en cuanto a Deep Learning y Machine Learning, referentes a la etapa de investigación realizada antes de empezar con el desarrollo de la red neuronal que se obtuvo con este trabajo.

Posteriormente se desarrolla de forma general el proceso de programación y diseño de la red neuronal en computación cuántica además de mostrar los resultados obtenidos con dicho algoritmo cuántico. Finalmente se enuncian las conclusiones y retroalimentaciones generadas para una próxima versión mejoradade este trabajo.

1.2. OBJETIVOS

GENERAL

Desarrollar un algoritmo sencillo de inteligencia artificial basado en computación cuántica que permita generar una base para futuros desarrollos de inteligencia artificial cuántica.

ESPECIFICOS

- Aprender y comprender las bases de inteligencia artificial y computación cuántica para ser aplicadas en un perceptrón cuántico.
- Conocer y usar la programación mediante Python en computación clásica para generar un código híbrido entre programación clásica y cuántica.
- Repasar y aplicar las librerías Qiskit de IMB para la programación del perceptrón cuántico.
- Programar un perceptrón sencillo en computación cuántica.

1.3. DESCRIPCIÓN DE LA ENTIDAD

1.3.1. INFORMACIÓN BÁSICA

Grupo Bernier es una empresa que nace en 2015 con el objetivo de cubrir las necesidades de las pequeñas y medianas empresas en cuanto a la sistematización y organización de la información, automatización de procesos, análisis de datos y facilidad de acceso a la información desde cualquier dispositivocon ayuda de la tecnología informática.

Gracias a la organización de la información mediante un software, se aumenta la velocidad y eficiencia de trabajo, de modo que se mejora la productividad de las empresas y se obtiene mayor tiempo para desempeñar otras actividades indispensables en la vida del trabajador.

Además del producto tecnológico que ofrece la empresa, también suministra una metodología de trabajo que rediseña y mejora los procesos; estableciendo estrategias organizacionales y mejora la rentabilidad de los procesos de calidad. No solo ofrece una solución tecnológica, sino que se involucra en las empresas desus clientes para ayudarles a mejorar e innovar.

Así, haciendo uso de la tecnología se ayuda a los clientes a ser empresarios organizados y eficientes, además de mejorar su calidad de vida.

1.3.2. SERVICIOS

1.3.2.1. INTELIGENCIA ARTIFICIAL

La inteligencia artificial es implementada con el fin de potencializar los procesos, la ejecución y capacidad de trabajo, mejorando la competitividad, la eficiencia y consecuentemente la rentabilidad de aquellas empresas que adquieran los servicios de Grupo Bernier y calidad de vida de sus integrantes.

1.3.2.2. CHATBOTS Y BOTS

Es un Sistema de interacción empresa-cliente en tiempo real sin recurso humano que mejora el tiempo de respuesta y amplía el campo de acceso a nuevosclientes. Así mismo se puede redireccionar el trabajo de algunos empleados a actividades de igual importancia en otras áreas de la empresa, además tiene las siguientes ventajas:

- Atención a múltiples usuarios al mismo tiempo manteniendo la calidad de servicio.
- Documentación automática en tiempo real
- Generación de estadísticas
- Automatización de actividades críticas en la empresa
- Atención las 24 horas
- Posibilidad de atender solicitudes en varios idiomas

1.3.2.3. ANALISIS PREDICTIVO DE DATOS

Consta de sistemas inteligentes que aprenden de sus propios datos con ayuda de procesos matemáticos y computacionales, con el fin de ayudar a las empresas clientes a aprender de su propio proceso y mejorar cada vez para ser más competitiva y eficiente.

1.3.2.4. INTELIGENCIA ARTIFICIAL

Se refiere a un apoyo virtual que se mantiene las 24 horas y los siete días de la semana en tiempo real con el fin de mejorar la capacidad de ejecución, disminuir el rango de error y seguir una ejecución más organizada gracias al retro aprendizaje y mejora en cada ejecución de las actividades estipuladas.

1.3.2.5. MARKET EYE

Con el fin de recolectar los datos de determinadas variantes con mayor dificultad de medición y procesar e interpretar lo que ocurre en tiempo real en el espacio de trabajo se plantea el uso de dispositivos de video que abren nuevas oportunidades de innovación y crecimiento en las empresas cliente. (Bernier G. , Grupo Bernier SAS, s.f.)

1.3.2.6. FERMIONS SOFTWARE'S

Se refiere a una herramienta para hacer sostenible el modelo de excelencia empresarial PYMES, es una herramienta que se encuentra disponible en la nube que ayuda a simplificar la operación del modelo de negocio de las empresas clientes, enfocándose en lo que realmente aporta valor sin tener que pensar en corregir errores.

Por otra parte, se especializa también en un software aplicable a laboratorios de ensayo que están en el proceso de implementación o ya tienen la norma ISO 17025 de 2005. (Bernier J. P., 2017)

1.3.2.7. TRANSFORMACIÓN DIGITAL

Integra la inteligencia artificial para mejorar los procesos de todo lo que tiene que ver con los recursos clientes de la organización con ayuda de la adquisición del conocimiento generado por la empresa, asignación de valor al conocimiento adquirido desde las personas los procesos y los productos.

La mejora de este proceso con ayuda de la inteligencia artificial dará como resultado una innovación que hará avanzar y optimizar la empresa y la producción de esta. Además, produce una diferenciación del servicio de la empresa con respecto a los competidores del mercado, esto genera un crecimiento exponencial de la empresa en todos los aspectos.

En conclusión, la transformación digital ayuda a identificar el conocimiento referente a la empresa, buscar una forma de aprovecharlo y realizar los procesos para usar esta información y que la empresa sea optimizada en su proceso de producción y crecer en el mercado. (Bernier G. , Transformacion digital, 2018)

1.3.3. CONTACTO CON EL JEFE INMEDIATO

Nombre: Jean Phillip Bernier

Cargo: jefe de innovación y cofundador de la empresa

Correo de contacto: jean.bernier@berniergroup.com

LinkedIn: linkedin.com/in/jean-phillip-bernier

2. DESARROLLO DE LA PRÁCTICA

La práctica académica se basó principalmente en investigación y desarrollo de programación en computación cuántica para la empresa Grupo Bernier SAS, inicialmente se realizó una etapa de investigación de las temáticas importantes para el desarrollo de la practica académica de forma eficiente. Para dicho desarrollo se tuvo en cuenta temas como computación cuántica, inteligencia artificial y programación en Python. A continuación, se muestra un resumen de dicha investigación, sin embargo, en su momento se vieron algunos temas como programación en Visual Studio, desarrollo por medio de las librerías cuánticas de Microsoft, temas en lo que no se hará mucho énfasis en el marco conceptual ya que no aportaron en gran medida al desarrollo del proyecto que se manejó para la práctica académica, aunque si se hará una breve descripción.

2.1. MARCO CONCEPTUAL

2.1.1. INTELIGENCIA ARTIFICIAL

La inteligencia artificial trata de solucionar el problema de construir una máquina que se comporte de forma similar al ser humano. De modo que pueda cumplir los siguientes aspectos:

- Actuar como personas: La forma de evaluar las condiciones y programas referentes en el algoritmo debe ser similar a las reacciones y forma de pensar humana.
- Razonar como personas: La parte indispensable para ser considerada inteligencia artificial es la forma en que se razona y se entiende la información suministrada al algoritmo, mas no el resultado como tal del proceso. Así, la ciencia cognitiva utiliza el principio de que el programa razona como la mente humana.
- Razonar racionalmente: Se hace uso de la lógica con el fin de mostrar un estudio y procesamiento de datos de forma racional.
- Actuar racionalmente: Se evalúa de forma objetiva los resultados que se quieren obtener con ayuda del software, así los pasos a seguir para completar el proceso son cuidadosamente evaluados para completar la tarea de forma más eficiente y con lógica racional.

Es importante tener en cuenta que los temas en los que se enfoca la inteligencia artificial son: el aprendizaje automático, la resolución de problemas y búsqueda, la

inteligencia artificial distribuida, la representación del conocimiento y sistemas basados en este. Además, la inteligencia artificial hace uso del lenguaje natural, la visión artificial, la robótica y el reconocimiento de voz, entre otros. (Torra, 2011)

2.1.1.1. HISTORIA

Los primeros referentes en cuanto a inteligencia artificial se remontan a los años 30 con el considerado padre de la inteligencia artificial Alan Turing. El inicio real se estima al año 1950, año en el cual Turing publica un artículo en la revista 'Mind', con el título: "Computing machinery and intelligence". En dicha publicación se cuestionaba si las máquinas pueden pensar y propuso un método para determinar dicha teoría. Actualmente sigue vigente el test de Turing que está basado en el experimento planteado en dicho artículo.

Otros investigadores consideran como punto de partida el año 1956, cuando, John McCarty, Marvin Misky y Claude Shannon mencionaron formalmente el termino durante la conferencia Dartmouth Summer Research Project on Artificial Intelligence., como: "la ciencia e ingenio de hacer máquinas inteligentes, especialmente programas de cálculo inteligente".

Otras fechas importantes en la historia de la IA son:

- 1997: El supercomputador Deep Blue de IBM, ganó el campeonato mundial de ajedrez Gari Kasparov. Es muy importante ya que ayudó a que la IA empezará a ser conocida.
- 2011: En este año el supercomputador Watson de IBM, gana el concurso televisivo Jeopardy en estados unidos, en el cual se realizan preguntas de cultura y conocimiento. Watson es una computadora que implementa el aprendizaje automático, y puede interactuar en lenguaje natural de modo que aprende con cada interacción con el usuario.
  Otro apunte importante es la presentación por Apple de su asistente virtual SIRI, donde se vieron las primeras muestras de aprendizaje profundo y automático.
- 2012: Google presentó su asistente virtual, Google Now, mientras que en 2014 Microsoft presentó a Cortana.
- 2016: El software de IA de Google Alpha Go se enfrentó al surcoreano Se-Dol, campeón de un juego milenario de estrategia llamado Go. El software ganó las tres primeras partidas.

- 2017: Algoritmo Libratus de IA creado por la universidad Carnegie Mellon venció a 4 de los mejores jugadores de póquer en un casino de estadosunidos. Dicha competición se realizó mano amano entre los jugadores y la máquina donde Alpha Go ganó por gran diferencia. (Empresarial, 2017)

2.1.1.2. PRINCIPIOS BÁSICOS

La inteligencia artificial se basa en ciertas metodologías de aplicación, dentro de ellas se encuentran los lenguajes de programación lógica, metodologías que se muestran a continuación:

- Sistemas expertos: Se refiere a softwares que emulan el comportamiento de un humano experto en la solución de determinado problema. Funciona con lógica proporcional y mejora con el tiempo.
- Metodología del simbolismo: está basada en la cognición computacional. Su propósito es entrar en una fase de investigación científica regular teniendo en cuenta el estudio de la cognición humana. Tiene en cuenta el pensamiento humano para la computación y asume prácticas de programación simulando el simbolismo humano.
- Metodología del conexionismo: tiene en cuenta los modelos de redes neuronales como el modelo de Hopfield, backpropagation y los algoritmos genéticos. El primero de estos se basa en el sistema nervioso, cada neuronade esta red se conecta desde su salida a otras neuronas a partir de conexionesy recibe así mismo, información de otras neuronas; es una conexión de una sola capa y un sistema de retroalimentación con componentes no lineales. La segunda se basa en propagación de forma que las señales de entrada son procesadas por una capa y entrega la información a la siguiente llamada capa oculta que puede ser una o varias quienes finalmente entregan la información ala capa de salida. Cada neurona de salida solo afecta el estado de la siguiente capa. Además, para el entrenamiento de la red neuronal se tiene en cuentaque se hace hacia atrás, empezando por la capa de salida y terminando con la capa de entrada, modificando la señal de error en cada capa hasta encontrar larespuesta más optima al problema planteado. Y por último los algoritmosgenéticos, quienes requieren para su aplicación de representaciones codificadas como un cromosoma. Así, cada cromosoma tiene varios genes quecorresponden a los parámetros del problema en cuestión y se representa por una cadena de genes N.

- Metodología del conductismo: en esta metodología se tienen en cuenta la lógica difusa que se basa en entender los cuantificadores del lenguaje de modo que para cada conjunto difuso debe existir una función de pertenencia para sus elementos que muestra que porcentaje de participación tiene el elemento en el conjunto. También se tiene en cuenta el control inteligente que se aplica a robótica en aspectos como los sistemas sensoriales, control de movimientos de articulaciones, planificación de métodos de implementación de secuencias y funcionamiento del robot. (Peyrna, 2018)

Algunas otras de las principales técnicas de la inteligencia artificial son:

Machine Learning: aprendizaje automático.

Lógica Difusa: toma valores al azar los contextualiza y los relaciona entre sí.

Vida Artificial: representa cualidades propias de seres vivos en ambientes de simulación.

Minería de Datos: extrae de forma discriminada la información necesaria a partir de las bases de datos para determinado proceso sondeando, preparando y explorando los datos para seleccionar la información que se pueda sacar de ellos.

Redes Bayesianas: son un modelo de probabilismo multivariado, el cual relaciona las variables de tipo aleatorio para indicar que influencia tienen estas en el proceso que se desarrolla.

Sistemas basados en reglas: son sistemas de representación del conocimiento que usualmente se usan en escenarios en los que el conocimiento que se desea explicar se puede representar con una serie de reglas.

Entre otras herramientas que se pueden utilizar para la implementación de la inteligencia artificial en los diferentes ámbitos de la industria. (APD, 2019)

2.1.1.3. ¿PARA QUE SIRVE?

La inteligencia artificial se aplica en muchos sectores de la industria, aquí se realiza un resumen de algunas de estas áreas de aplicación:

- Salud y biotecnología: Es de gran importancia tener un diagnóstico preciso y lo más eficiente posible, aquí es donde entra la inteligencia artificial a acelerar el proceso de análisis tanto de sintomatología como de aspectos genéticos que pueden llevar a desarrollar determinadas enfermedades

- Comercio: Uno de los usos principales es el marketing y comercialización de productos, se utiliza para hacer compras en línea, de modo que la inteligencia artificial ofrece pronósticos de ventas y mejora la eficiencia de servicio alcliente.
- Educación: Un simulador inteligente puede detectar ciertos comportamientos en el proceso de aprendizaje de un estudiante, además de ofrecerle las herramientas específicas para mejorar en sus falencias y así aportar a su aprendizaje de forma más eficiente e interactiva.
- Servicios financieros: en cuanto a las empresas financieras en importante mantener estudios de riesgos y suponer patrones de mercado que ayuden a prever los efectos que se puedan dar por determinados comportamientos en la cartera de clientes y/o proveedores. Así de forma automática genera estudios y recomendaciones para ayudar a mantener la producción de la empresa. (Next, 2019)
- Existen muchas otras aplicaciones, las anteriormente mencionadas son solo para dar una idea de lo importante que es la inteligencia artificial y que se puede utilizar en todos los ámbitos de la industria.

2.1.1.4. REDES NEURONALES

En el mudo tecnológico de hoy se viene desarrollando muchos algoritmos que incluyen inteligencia artificial. Una de las herramientas más importantes en el ámbito del Machine Learning son las redes neuronales, las cuales constan de un grupo de algoritmos que se diseñan especialmente para que determinen, reconozcan o detecten patrones a partir de un conjunto de datos. Una de sus mejores características es que entrenan a la computadora para resolver problemas con gran complejidad.

El funcionamiento de una red neuronal se basa principalmente en la idea de realizar actividades de reconocimiento y selección que realizaría una persona fácilmente. En pocas palabras buscar simular el funcionamiento de neuronasbiológicas para funcionar en forma interconectada y jerárquica para procesarinformación a partir de determinados datos.

Están organizadas en capas que se clasifican normalmente en tres tipos; la capa de entrada, las capas ocultas y la capa de salida. La primera es la capa de neuronas que recibe la información que se quiere procesar y estudiar inicialmente,

las capas ocultas son las que determinan la potencia de la red neuronal ya que entre más neuronas es estas capas y entre mayor sea el número de capas ocultasmayor capacidad tendrá la red neuronal, en estas capas se aplican los valor de pesos que van a modificar la red para responder de acuerdo con el uso que se le va a dar a esta.

Dichos pesos son las variables que se varían al momento de entrenar la red, teniendo en cuenta los valores de entrada y los valores de salida quienes tienen relación directa con la capa de salida antes mencionada. En esta capa se genera la solución final del algoritmo. La estructura más sencilla de una red neuronal se muestra en la Fig. 1. (Everywhere, 2018)

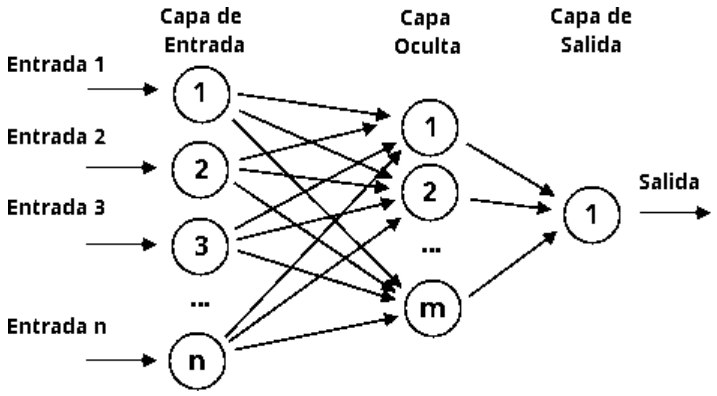

Fig. 1. Estructura general de la red neuronal. (innovation, 2019)

2.1.1.5. TIPOS DE REDES NEURONALES

Las redes neuronales se pueden clasificar según su topología o estructura de red donde se tienen en cuenta el número de capas el tipo de capas y la direccionalidad e las neuronas y según su algoritmo de aprendizaje, donde se tienen en cuenta como la red aprende, la forma de sus patrones, si es supervisada o no, si es competitiva, etc.

REDES MONOCAPA: Se refiere a aquellas redes neuronales diseñadas con una sola capa. Se crean conexiones laterales para poder realizar unión de las neuronas. Dentro de este grupo existes algunas que permiten la conexión de una neurona consigo misma, estas son llamadas auto recurrentes.

REDES MULTICAPA: Estas redes están compuestas por varias capas. Además, se pueden clasificar por la forma de conexión de sus neuronas, lo más común es

que las capas de la red neuronal se conecten en el orden que la señal viaja desde la entrada hacia la salida. Este tipo de conexión es denominada hacia a delante o Feedforward.

Por otra parte, se encuentran las redes con conexión hacia atrás, son llamadas así porque puede tener algunas conexiones en sentido contrario a la dirección de la señal de información, estas conexiones van de salida hacia entrada.

APRENDIZAJE SUPERVISADO: En este tipo de red se suministra las entradas y las salidas deseadas para el algoritmo y se aplica una fórmula matemática que minimiza el error ajustando los pesos para llegar lo más cerca posible a las salidas estipuladas para el entrenamiento. En general se sigue la siguiente secuencia para realizar todo el proceso de aprendizaje:

- Se inicializan los pesos de la red con valores aleatorios
- Se asignan los patrones de entrada y de acuerdo con los pesos estimados en el numeral anterior se genera una salida que puede no ser la correcta.
- Se calcula el error entre la salida del numeral anterior y la salida que se espera luego del entrenamiento.
- Se ajustan los pesos de la red para minimizar el error calculado en el numeral anterior
- En dado caso que el error sea mayor que cierto criterio de histéresis determinado previamente, se vuelve a ejecutar los numerales anteriores excepto el primero.

Entre las redes más utilizadas con aprendizaje supervisado se encuentran el perceptrón sencillo, perceptrón multicapa y la red Hopfield.

APRENDIZAJE NO SUPERVISADO: en este tipo de aprendizaje no se hace necesario asignar al algoritmo los patrones de salida, ya que el mismo produce patrones de salida consistentes. Entonces la red clasifica en categorías los patrones parecidos, esta similitud se determina con el proceso de entrenamiento, donde el error es el que determina a que categoría pertenece cada patrón.

La red realiza una comparación entre el patrón de entrada y los datos de entrenamiento almacenados, de modo que agrupa dichos datos de entrada en uno u otro grupo o categoría, aunque sea desconocida en principio la salida que corresponde al dato de entrada en cuestión ni los atributos del mismo. Así, la red de acuerdo con sus algoritmos de aprendizaje diferencia y clasifica los parones en determinadas categorías.

Dentro del aprendizaje no supervisado existen tres tipos: el aprendizaje por componentes principales el cual se basa en hallar características similares entre muchos patrones de entrada con ayuda de un pequeño número de neuronas, el aprendizaje competitivo donde las neuronas intentan representar una clase; la neurona que más se asemeja al patrón de entrada depende específicamente de la comparación de los pesos de esta con los de los patrones de entrada. El aprendizaje se basa en reforzar las conexiones para que la neurona seleccionada se parezca cada vez más al patrón de la clase en cuestión. Y por último se encuentra el aprendizaje reforzado, el cual tiene en cuenta la noción de condicionamiento por refuerzo; tal que se aprenden las conductas reforzadas de forma positiva y negativa premiando los pesos sinápticos cuando se acierta a la salida y penalizándolos cuando se falla.

Dentro de los principales tipos de redes neuronales se encuentran:

- El Perceptrón Simple.
- La Red de Hopfield.
- El Perceptrón Multicapa.
- Red neuronal Competitiva Simple.
- Redes Neuronales Online ART1.
- Redes neuronales competitivas ART2.
- Redes neuronales autoorganizadas: Mapas de Kohonen. (Ballesteros, s.f.)

Mas adelante se hará referencia a dos de ellos que fueron de vital importancia para el desarrollo de la práctica.

2.1.1.6. ¿PARA QUE SIRVEN?

Existen muchos tipos de redes neuronales y cada una tiene un tipo de aplicación particularmente más adecuado, dentro de las principales aplicaciones comerciales podemos destacar:

- Dentro del área de la biología:
- Obtención de modelos de retina
- Estudio del cerebro y otros sistemas.
- En cuanto a medio ambiente:
- Prevé el tiempo y el clima.
- Analiza tendencias y patrones
- En el ámbito empresarial:

- Explora bases de datos
- Optimiza el flujo del tránsito controlando la temporización de semáforos y organiza los horarios de vuelo y plazas de aviones.
- Reconocimiento de caracteres escritos
- Modelado de sistemas para automatización y control.

• En el área financiera:

- Estudios de evolución de precios
- Identificación de falsificaciones y valoración de riesgos en los créditos
- Interpretación y reconocimiento de firmas

• En manufactura:

- Control de líneas de producción
- Inspección de calidad
- Automatización de robot y sistemas de control con uso de visión artificial y sistemas sensoriales.
- Entre muchas otras aplicaciones.

• En medicina:

- Analizadores de sonidos para ayudar a audición de personas sordas.
- Diagnóstico y tratamiento de enfermedades a partir de sintomatología y datos producto de exámenes médicos.
- Monitoreo y predicción de reacciones en cuanto a cirugías y suplemento de medicamentos.
- Análisis de posibles causas para ciertas afecciones.

En su mayoría la inteligencia artificial se aplica en reconocimiento de patrones, clasificación y corrección y/o reconstrucción de señales a partir de valores parciales. (Matich, 2001)

2.1.1.7. PERCEPTRÓN

El perceptrón monocapa está compuesto simplemente por una capa de entrada y una de salida. Así, la única neurona presente que se encuentra en la capa de salida toma los datos de entrada, los procesa con el valor de peso y umbral para aplicarle una función de transferencia de tipo escalón.

Así, la principal función del perceptrón simple es el de clasificar los datos de entrada en dos tipos, como se evidencia en el Fig. 2.

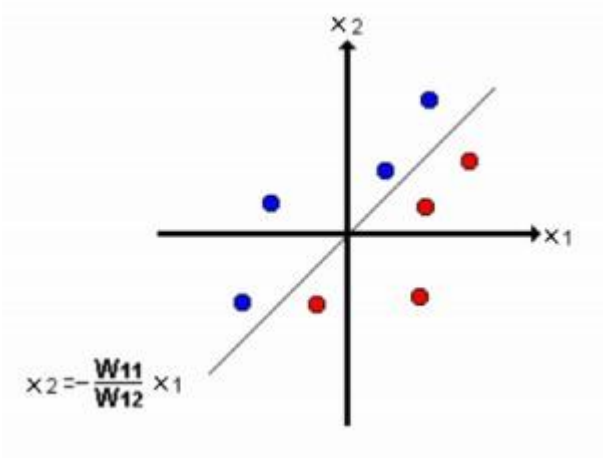

Fig. 2. Clasificación con el perceptrón sencillo. (Bibing, s.f.)

Definición Formal

Definimos " perceptrón simple" como un dispositivo de computación con umbral ***U y n*** entradas reales $x_1,\ldots, x_n$
a través de arcos con pesos $w_1,\ldots, w_n$ y que tiene salida 1 cuando $\Sigma_i w_i x_i \geq U$ y 0 en caso contrario.

Fig. 3. Definición del perceptrón, tomada textualmente. (Bibing, s.f.)

Las entradas pueden proceder de otros perceptrones o de otro tipo de unidades de computación. La diferencia entre las entradas que generan una salida 1 con respecto a las que genera salida 0 o -1 según sea el caso está dada por un hiperplano que divide el espacio de entradas en dos semiplanos.

El hecho que una de las posibles salidas puede ser de tipo 0 o -1 radica en que el perceptrón puede tener dos tipos de funciones de transferencia, escalón consalidas 1, -1 o escalos con salidas 1,0; su uso depende del valor de entrada que se espera obtener de la red, es decir si esta es unipolar o bipolar.

No todas las funciones se pueden representar con ayuda de un perceptrón sencillo. Solo aquellas de se pueden separar sus valores de clase 1 de los declase 0 con ayuda de un hiperplano, de no ser así se hace necesario aplicar más neuronas a la red para lograr el estudio del sistema.

APRENDIZAJE DE LA RED:

Como todas las entradas están con varias salidas, entonces hay que revisar la separabilidad lineal para cada salida. El algoritmo supervisado del perceptrón usa n valores que se pueden clasificar en dos tipos en el espacio extendido $\boldsymbol{n + 1}$ dimensional. Se considera el umbral como una entrada constante de valor 1 y buscamos un vector de pesos capaz de separar completamente ambos conjuntos de entradas, para ello se sigue el siguiente algoritmo:

- Fijar los pesos para las entradas
- Seleccionar un elemento de entrenamiento (entrada, salida).
- Introducir la entrada seleccionada en la red para calcular su salida Y
- Comparar la salida Y con la salida esperada
  Para $\boldsymbol{j = 1 \ldots q}$

Si salida j es incorrecta, entonces; $\boldsymbol{W'ij = Wij - (dj - Yj)\, Xi}$

Si la salida (j) no correcta, Si $\boldsymbol{Yj = 0}$, aplica: $\boldsymbol{W'ij = Wij + Xi}$]

Si $\boldsymbol{Yj = 1}$, entonces $\boldsymbol{W'ij = Wij - Xi}$

- Se ejecuta los numerales 2, 3 y 4 hasta que todas las salidas reales sean iguales a las esperadas. (Bibing, s.f.)

2.1.1.8. PERCEPTRÓN MULTICAPA

El perceptrón monocapa no es lo suficientemente potente, por tanto, se hace necesario el uso de una red más compleja con mayor número de neuronas y capas.

Primero que todo es de vital importancia saber que es una capa. Una capa es un grupo de neuronas que se encuentra dentro de la red en un mismo nivel pero que no se interconectan entre sí. Así el perceptrón multicapa no es más que un grupo de perceptrones sencillos que logran realizar tareas mucho más complejas y con mucha mayor exactitud.

La representación del perceptrón multicapa se muestra en la Fig. 2.

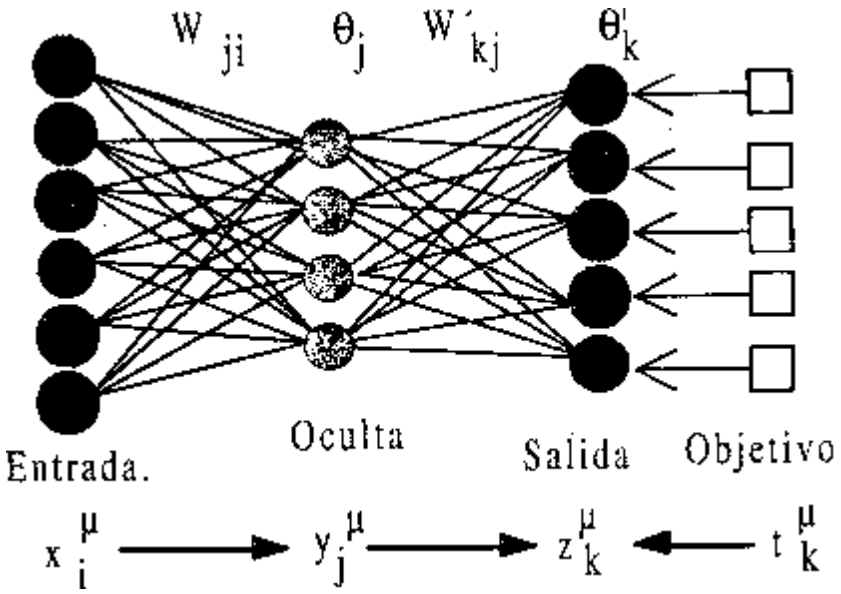

Fig. 4. Estructura del perceptrón multicapa. (Palacios, s.f.)

Las neuronas de la capa oculta se propagan de acuerdo con la suma ponderada de los pesos con las entradas y a este resultado se le aplica una función sigmoide la cual acota la respuesta.

FUNCIÓN SIGMOIDE: Para el caso de la red multicapa, usualmente se usa la retro propagación y para realizar el entrenamiento de la red se debe tener en cuenta la función de coste global que normalmente se determina con el error cuadrático medio, donde se comparan los datos de entrenamiento y las salidas deseadas mediante la fórmula Ec. 1.

$$E(w_{ij}, \theta_j, w'_{kj}, \theta'_k) = \frac{1}{2} \sum_p \sum_k \left[d_k^p - f\left(\sum_j w'_{kj} y_j^p - \theta'_k \right) \right]^2$$

Ec. 1. Fórmula para calcular el error cuadrático medio. (Palacios, s.f.)

Luego de tener el valor del coste global se deben calcular los valores delta que refiere a cuanto debe variar los pesos para que se mejore el error de la respuesta de la red neuronal respecto a los datos de entrenamiento. Para esto se tiene en cuenta la ecuación Ec.2.

$$\delta w'_{kj} = -\epsilon \frac{\partial E}{\partial w'_{kj}}$$

$$\delta w'_{ji} = -\epsilon \frac{\partial E}{\partial w_{ji}}$$

$$\delta w'_{kj} = \epsilon \sum_p \Delta'^p{}_k y^p_j \qquad con \qquad \Delta'^p{}_k = [d^p_k - f(v'^p{}_k)] \frac{\partial f(v'^p{}_k)}{\partial v'^p{}_k}$$

$$\delta w_{ij} = \epsilon \sum_p \Delta^p_j x^p_i \qquad con \qquad \Delta^p_j = \left(\sum_k \Delta'^p{}_k w'_{kj} \right) \frac{\partial f(v^p_j)}{\partial v^p_j}$$

Ec. 2. Ecuaciones para el cálculo de los deltas. (Palacios, s.f.)

Siendo $\boldsymbol{Y_k}$ las salidas de la capa.

Así la secuencia general del funcionamiento del entrenamiento por backpropagation sería así:

- Inicialización de pesos y umbrales de forma aleatoria.
- Para cada patrón de entrenamiento se obtiene la respuesta de la red ejecutando el algoritmo hacia adelante normal de la red neuronal.
- Se calculan los errores asociados a cada entrada y salida según la ecuación Ec. 1.
- Se calcula el incremento para cada uno de los pesos y umbrales según las ecuaciones mostradas en Ec. 2.
- Se actualizan los pesos y umbrales para volver a calcular el error y repetir el proceso hasta llegar a un error menor al criterio que se desee. (Palacios, s.f.).

2.1.2. COMPUTACIÓN CUÁNTICA

Es una analogía de la computación clásica, pero con mucha mayor potencia, está basada en el uso de qubits en lugar de bits y da lugar a compuertas cuánticas que trabajan de forma similar a las clásicas, pero con mayor capacidad y eficiencia.

La computación clásica se basa principalmente en el modelo de estados del átomo para realizar los procesos en vez del lenguaje binario que se usa en las computadoras clásicas.

Para usar este tipo de estados se tiene en cuenta el principio de superposición cuántica que principalmente sigue la teoría de Schrödinger. (Frutos, 2017)

2.1.2.1. HISTORIA

El origen de la teoría cuántica refiere al periodo desde 1900 a 1905, por Max Planck y Albert Einstein, quienes siguieron la idea de Ludwig Boltzmann planteadaen 1877, sobre la entropía y la probabilidad. Su estudio se basó en resolver el problema planteado por la radiación del cuerpo negro.

En diciembre de 1900, Planck mencionó por primera vez el concepto de cuanto de energía dentro de sus investigaciones con el fin de describir las propiedades espectrales de la radiación. Las teorías y descubrimientos de Planck fueron el primer paso para la introducción del mundo a la era cuántica.

Por otra parte, Einstein propuso modificar la teoría clásica del campo electromagnético de forma que recurrió a un enfoque mecánico estadístico. En 1905, sugirió cuales debían ser las constituyentes elementales de la radiación y cuales debían ser sus procesos dentro de la aproximación mecánico estadística. Einstein también descubrió que la luz no es una onda continua, sino que a veces tienen un comportamiento similar al de una partícula, lo cual es llamado dualidad onda partícula. Así, Einstein se dio cuenta que la luz se comporta como si viniera en pedazos, en cuantos que para el caso de la luz son llamados fotones.

En 1923 Niels Bohr comenzó a complementar el modelo cuántico explicando la estructura del átomo mediante la mecánica cuántica.

En 1925, Erwin Schrödinger formulo la ecuación con su nombre, la cual explica todos los fenómenos de cuánticos anteriores y da una forma sistematizada de explorar el mundo atómico.

“En una teoría completa, cada elemento corresponde a un elemento de la realidad, una condición suficiente para que una cantidad física sea real, es la posibilidad de predecirla con certeza sin interrumpir el sistema. Según la mecánica cuántica es posible que dos partículas estén tan estrechamente entrelazadas que formen un sistema único en el que ninguno de ellas tenga un estado cuántico propio (Schlegel, 1971).”

La segunda revolución cuántica cuenta con dos características principales: la capacidad de controlar la rareza del mundo cuántico incluidas la superposición y el entrelazamiento, y aquí es donde llega la era de la información, donde se hace ver la información cuántica.

En 1985 el físico americano Richard Feynman, publico un artículo donde abordaba la posibilidad de implementar principios cuánticos para el funcionamiento de computadoras. Aquí introdujo señales, patrones, partículas hipotéticas localizadas en el núcleo atómico.

Stephen Hawking “los ordenadores cuánticos se basan en el hecho de que el estado cuántico de la memoria de un ordenador contiene mucha más información que sus descripciones clásicas” (Abbott et al., 1999).” (Álvarez, 2016)

2.1.2.2. ¿PARA QUE SIRVE?

En los últimos años algunas empresas aliadas con IBM, invierten en la investigación de computación cuántica para tener el privilegio de ser los primeros en utilizarla en el futuro.

Uno de los posibles usos que se le pueden dar a la computación cuántica es el diseño de materiales, es decir en la idea de descubrir mejores materiales para la construcción de componentes tecnológicos.

Otra área es en la optimización de procesos financieros, como el modelo de riesgo financiero, como la capacidad de cómputo exigida para estos procesos es muy alta, el uso de la computación cuántica mejoraría la eficiencia en dichos procesos.

Otra industria bastante beneficiada es la farmacéutica, ya que se puede utilizar en la simulación de moléculas químicas para la producción de nuevos medicamentos. "Hoy día es casi imposible simular la molécula de agua, que son dos átomos de hidrógeno y uno de oxígeno, muy simple. Pero es algo que las computadoras de hoy día no pueden simular. Esperamos que más con esta tecnología podamos simular moléculas más complicadas para el desarrollo de fórmulas químicas". (Ortega, 2019)

Algunas aplicaciones ya implementadas son:

> La criptografía cuántica, ya se ha utilizado a alto nivel, para asegurar los resultados electorales en Suiza, allí se desarrolló la tecnología básica para la criptografía cuántica.

- Tele transportación cuántica. La tele transportación cuántica no tiene nada que ver con lo que se ve en ciencia ficción, porque no se traslada ninguna materia de un punto A un punto B, es la información cuántica lo que se traslada de A hacia B.
- Simulación de sistemas cuánticos
- Diseño de nuevos super conductores para dirigir trenes de levitación magnética.
- Identificación de mercurio en el pescado.
- Detectar bombas
- Producción de sensores más sólidos, precisos y sensibles.
- Codificación de información con mayor grado de seguridad. (Álvarez, 2016)

2.1.2.3. PRINCIPIOS BÁSICOS

Primero que todo es importante tener en cuenta que la computación cuántica en su forma más elemental se puede comparar con la computación clásica, ya que en cuanto a su unidad básica de información (bits) tiene un paralelo cuántico que es el qubit. La diferencia más importante entre estos dos tipos de unidades de información es que el bit clásico solo puede almacenar uno de dos estadosposibles, sin embargo, por el lado de los bits cuánticos o qubits también semuestra un estado, pero dicho estado puede estar en múltiples valores gracias el principio de superposición. Lo más importante para tener en cuenta con computación cuántica es lo siguiente:

- Vemos $|\mathbf{0}>$ y $|\mathbf{1}>$ como los vectores $\binom{\mathbf{1}}{\mathbf{0}}$ y $\binom{\mathbf{0}}{\mathbf{1}}$ respectivamente respecto a una base de $\boldsymbol{C^2}$.
- Un estado de superposición es una combinación lineal $a_0\binom{\mathbf{1}}{\mathbf{0}} + a_1\binom{\mathbf{0}}{\mathbf{1}}$

  donde a_0 y a_0 pertenecen a $\boldsymbol{C}$ tales que $|a_0|^2 + |a_1|^2 = \mathbf{1}$
- A a_0 y a_1 los llamamos amplitud de $|\mathbf{0}>$ y $|\mathbf{1}>$ respectivamente.

Extensión de la notación a varios qubits: los estados básicos se denotan por $|\mathbf{0} \ldots \ldots \ldots \mathbf{0}>$, $|\mathbf{0} \ldots \ldots \ldots \mathbf{1}>$, hasta $|\mathbf{1} \ldots \ldots \ldots \mathbf{1}>$ donde existen $\boldsymbol{2^n}$ estados básicos con $\boldsymbol{n}$ qubits. Cada estado básico corresponde a un vector de la base de $\boldsymbol{C^{n^2}}$ dado por:

$$|a_2, n> = \begin{matrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{matrix} <--a$$

Ec. 3. Representación vectorial del estado cuántico

Ventor con tamaño n.

Los estados en superposición son combinaciones lineales de la forma: $a_0 |0 \dots 1> + \cdots + | a_{2^n} |1 \dots 1>$ con a_n perteneciente a C.

Tales que,

$$|a_0|^2 + |a_{2^n-1}|^2 = 1$$

Decimos que es un conjunto de qubits está en estado producto si su estado puede ponerse como producto de los estados de sus componentes. En caso contrario, si no se puede, entonces se dice que está en un estado de entrelazamiento.

Algunos de los estados más importantes en computación cuántica son:

$$|+> = \frac{1}{\sqrt{2}}| 0> + \frac{1}{\sqrt{2}} |1>$$

Ec. 4. Representación del estado cuántico |+>

$$|-> = \frac{1}{\sqrt{2}}| 0> - \frac{1}{\sqrt{2}} |1>$$

Ec. 5. Representación del estado cuentico |->

$$\frac{|00> + |11>}{\sqrt{2}}$$

Ec. 6. Representación del par EPR o estado de Bell. (Sáenz, 2019)

2.1.2.4. PRINCIPIO DE SUPERPOSICIÓN

Gracias a la mecánica cuántica, se demostró que existen partículas que pueden estar en dos lugares a la vez, y una de las hipótesis plantea que estas partículas pueden viajar entre dos universos paralelos.

Pese a que puede verse inverosímil, este principio está demostrado y se conoce

como la propiedad de la SUPERPOSICIÓN CUÁNTICA.

Los estudios de mecánica cuántica, han probado que las micropartículas que componen el cosmos están en todos los estados posibles hasta el momento en que el observador interviene.

El principio de superposición nos permite suponer que las partículas pueden estar en varias realidades a la vez o de aparecer y desaparecer constantemente, lo que abre la posibilidad de que haya una cantidad indefinida de universos paralelos o el hecho de que haya partículas que puedan ser y no ser a la vez.

El experimento imaginario de Erwin Schrödinger explica el principio de superposición; planteando un gato metido en una caja opaca y cerrada dentro de la cual se encontraba una botella de gas venenoso con un dispositivo de activación. Según Schrödinger el felino tenía las mismas posibilidades de estarvivo o muerto. Así, ya que la mecánica cuántica demuestra que las partículas pueden tomar diferentes estados al mismo tiempo, el gato podía estar vivo y muerto al mismo tiempo, hasta el instante en que el observador entrada en escenay las partículas se ven forzadas a tomar uno de los dos estados posibles.

Para cumplir el estado de superposición existe otra teoría suponiendo que solo sea posible una realidad. En este caso el comportamiento de las partículas no se debería a sus viajes ente universos sino a la propiedad que poseen de asumir diversos estados en un mismo universo. Así, las partículas toman todos los estados posibles hasta el momento en que el observador entra en escena, entonces las partículas no viajan, sino que se modifican a sí mismas constantemente hasta el momento en que son observadas. (Roselló, 2017)

2.1.2.5. PUERTAS CUÁNTICAS

Para manipular la información almacenada en un conjunto de qubits, se usan compuertas cuánticas. Para definir una puerta cuántica se necesita definir su efecto sobre los estados básicos y exigir que actúe linealmente sobre los estados de superposición, además de ser unitaria.

Dentro de las puertas cuánticas más importantes se encuentran:

Puerta X

La puerta cuántica X, convierte el estado $|\mathbf{0}>$ en estado $|\mathbf{1}>$ y viceversa. Es una puerta lineal, de forma que $a_0|\mathbf{0}> + a_1|\mathbf{1}> \leftrightarrow a_0|\mathbf{1}> + a_1|\mathbf{0}>$. Además, es una puerta unitaria.

Su notación matricial es las siguiente:

$$\boldsymbol{X} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} \end{bmatrix}$$

Ec. 7. Notación matricial, puerta X

Puerta V

La puerta V añade una fase relativa i al estado $|\mathbf{1}>$, dejando el estado $|\mathbf{0}>$ inalterado. Equivale a una rotación de $\frac{\boldsymbol{\pi}}{\mathbf{2}}$ alrededor del eje Z.

La notación de forma matricial de dicha puerta es:

$$\boldsymbol{V} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{i} \end{bmatrix}$$

Ec. 8. Notación matricial de la puerta V.

Puerta de Hadamard.

La puerta de Hadamard es usada principalmente para generar superposiciones a partir de los estados de la base computacional: mapea el estado $|\mathbf{0}>$ a $\frac{|\mathbf{0}>+|\mathbf{1}>}{\sqrt{\mathbf{2}}}$, y el $|\mathbf{1}>$ a $\frac{|\mathbf{0}>-|\mathbf{1}>}{\sqrt{\mathbf{2}}}$. Equivale a una rotación de π alrededor del eje XZ.

La notación matricial es:

$$\boldsymbol{H} = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}} * \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & -\mathbf{1} \end{bmatrix}$$

Ec. 9. Representación matricial de la puerta Hadamard.

Puerta Z

La puerta Z mapea el estado $|\mathbf{1}>$ al $-|\mathbf{1}>$, dejando $|\mathbf{0}>$ inalterado. Equivale a una rotación de $\boldsymbol{\pi}$ alrededor del eje Z.

Su notación matricial refiere a la puerta V al cuadrado (V^2):

$$Z = V^2 = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Ec. 10. Representación matricial de la puerta Z.

Otra puerta importante es la CNOT, esta utiliza más de un qubit a la vez. También llamada control NOT, cuyo efecto es: envía $|a \quad b> \rightarrow |a \quad b \oplus a>$, donde a y b son

{0,1}.

Se puede describir su funcionamiento así:

El primer qubit se llama control, y el segundo se llama objetivo. Si el control es 0 el objetivo se mantiene, si control es 1 el objetivo es negado.

En notación matricial:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Ec. 11. Notación matricial de la puerta CNOT. (R, 2018)

2.1.3. PROGRAMACIÓN

El trabajo realizado en el informe actual consta principalmente de programación o diseño de un algoritmo que cumpla con el principio y la función principal de un perceptrón cuántico, por tal razón se hace necesario el estudio previo del lenguaje de programación en el software de desarrollo y las librerías que aplican el lenguaje cuántico. Consecuentemente dentro de la práctica académica se realizó el estudio de softwares como Visual Studio y Python con librerías cuánticas utilizadas en Microsoft (Q-Sharp) y en IBM (Qiskit). A continuación, se realizará un pequeño abrebocas de estos softwares y librerías, sin embargo, para obtener información más detallada el lector se puede dirigir a la bibliografía mencionada para cada uno de los numerales.

2.1.3.1. VISUAL STUDIO

Es un panel de inicio creativo que se puede utilizar para depurar, editar y compilar código y finalmente publicar una aplicación. Un entorno de desarrollo integradoque tiene características con gran cantidad de aplicaciones en el desarrollo de software.

En la figura 5 se muestra la interfaz principal de Visual Studio.

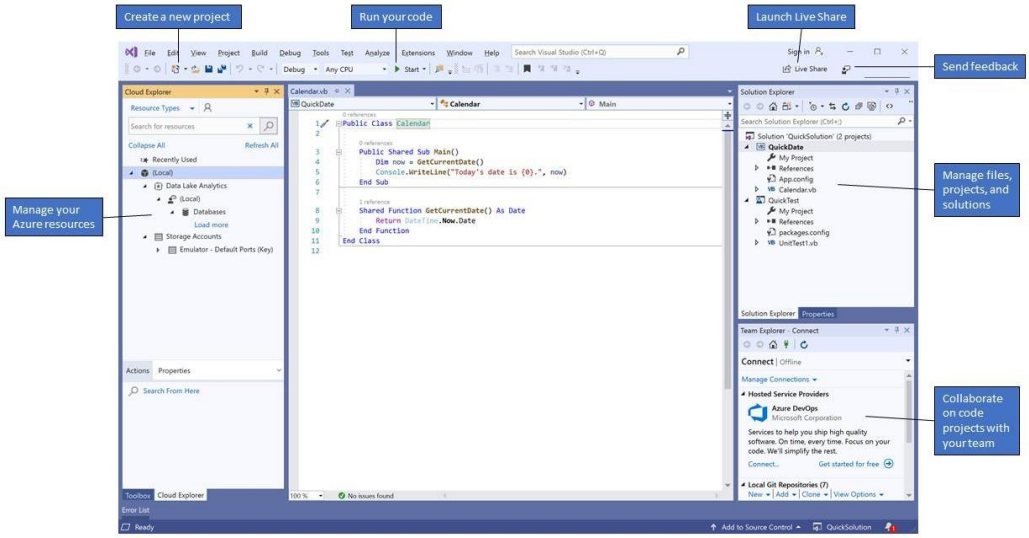

Fig. 5. Interfaz de Visual Studio. (Microsoft, 2019)

Además, de los depuradores y editores estándar Visual Studio proporciona la característica de compilación, herramientas de finalización de código, diseñadores gráficos y muchas más características para facilitar el proceso de desarrollo de software.

Algunas de las principales características de productividad que hacen destacar a Visual Studio son:

Los subrayados ondulados que son utilizados para alertar de errores o posibles problemas en el código a medida que se realiza la codificación. Al mantener el mouse sobre el subrayado ondulado se obtiene información adicional sobre el error además de proporcionar en algunos casos acciones rápidas para corregir dicho error.

La limpieza de código consta en que solo con un clic se puede dar formato al código y aplicar alguna corrección sugerida en la configuración de estilo delcódigo. Ayuda a solucionar problemas en el código antes de realizar la revisión de este.

La refactorización refiere a operaciones como el cambio de nombre de variables, la extracción de una o más líneas de código en métodos más eficientes, reordenamiento de los parámetros entre otras funciones.

IntelliSense: es un conjunto de característica que muestran información sobre el código en el editor y en ocasiones predice pequeños fragmentos de código. Es como tener una base de datos de los comandos necesarios para realizar laprogramación.

El cuadro de búsqueda es un apoyo para encontrar de forma rápida lo que se necesite dentro del entorno de Visual Studio, así, al realizar una búsqueda esta función proporciona un listado de opciones que se adaptan a lo que se está necesitando.

Live Share se usa para editar y depurar con conjunto con otros usuarios en tiempo real sin importar el lenguaje o tipo de aplicación que se esté manejando.

La jerarquía de llamadas es una ventana donde se muestran los diferentes métodos que llaman a un determinado método. Es útil en el caso de edición o eliminación de determinados métodos dentro del código.

CodeLens ayuda a buscar referencias, errores, cambios en el código, elementos de trabajo revisiones del código, o pruebas unitarias sin necesidad de salir del editor.

Ir a definición es una característica que ayuda a ir a la ubicación inicial de creación de una función o un tipo que han sido llamados en otra línea del código.

La ventana de ver la definición hace una vista previa de la función que se está llamado en el comando seleccionado.

Es importante tener en cuenta que en esta referente solo se mencionó de forma muy amplia la información referente al software, en caso de querer profundizar y aprender los comandos y la forma de aplicación de estos en la programación en los diferentes lenguajes que se manejan es recomendable dirigirse a la referencia bibliográfica de este apartado. (Microsoft, 2019)

2.1.3.2. PYTHON

Python es una herramienta de programación de propósito general. Es un lenguaje de scripting independiente y orientado a objetos con capacidad para realizar diferentes programas desde aplicaciones Windows a servidores o páginas web.

La interfaz principal de Python se muestra en la figura 6.

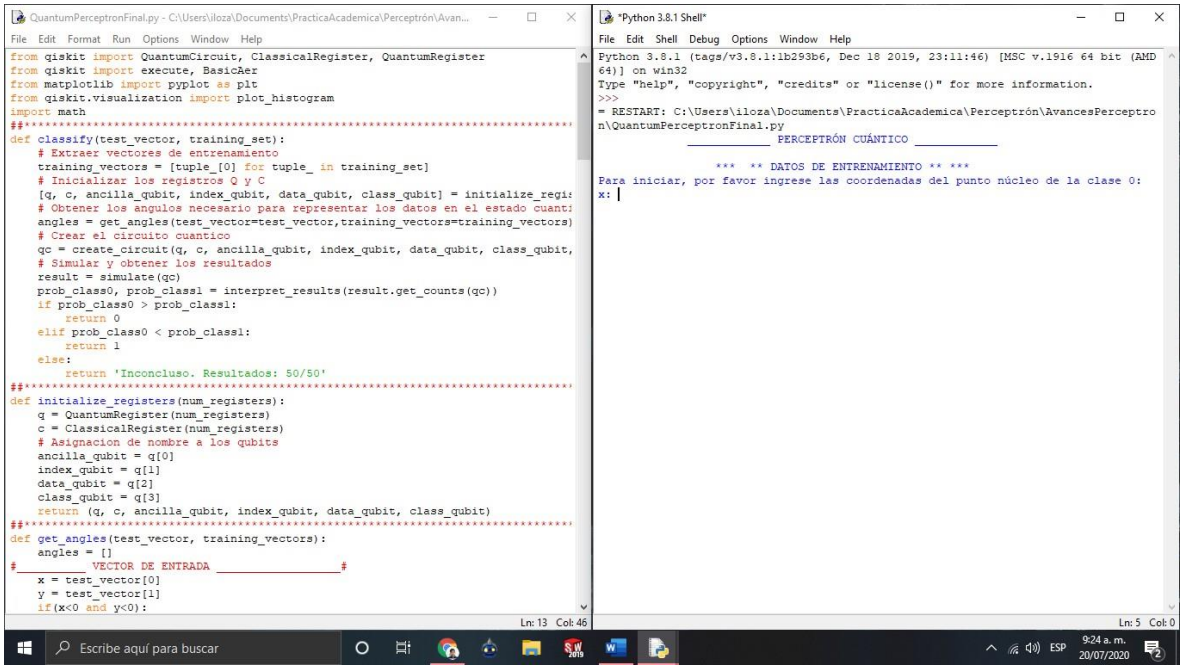

Fig. 6. Interfaz principal de Python.

Las principales características que componen el lenguaje de programación son las siguientes:

- Es interpretado, no compilado, usa un tipado dinámico.
- Es multiplataforma, así se puede ejecutar en diferentes sistemas operativos sin complicaciones.
- Es un lenguaje multiparadigma, es decir, soporta varios paradigmas de programación como estructurada, orientación a objetos, programación imperativa y programación funcional.
- El formato del código es estructural.

VENTAJAS:

- Simplificado y rápido: hace fácil la adaptación del usuario a un modo de lenguaje de programación. Python propone un patrón.
- Elegante y flexible: da muchas herramientas, no se hace necesario preocuparse tanto por los detalles, por ejemplo, no es necesario declarar cada tipo de dato si se quieren listas de varios tipos de datos.

- Producción sana y productiva: es sencillo de aprender, direccionando al uso perfecto de las reglas. Python ayuda a ser más productivo, a generar los programas en menor tiempo e influye en la idea de mejorar cada vez.
- Ordenado y limpio: es muy legible, cualquier otro programador puede entender un código gracias a la distribución y organización del lenguaje.
- Portable: la idea de baterías incluidas son las librerías más utilizadas quienes están incluidas en el intérprete y no es necesaria la instalación de estas.
- Comunidad: Una de las ventajas más importantes para el desarrollo es la comunidad para actualizaciones y resolución de problemas además todas las actualizaciones se hacen de forma democrática.

DESVENTAJAS:

- La curva de aprendizaje no es tan sencilla en la parte de programación web.
- La mayoría de servidores no tiene soporte a Python, y si lo soportan la configuración es difícil.
- Algunas librerías por defecto no son del gusto de la comunidad por lo que usualmente se usan librerías de terceros.

Para ver el apoyo o ayuda disponible sobre la sintaxis de Python, basta con ejecutar el comando

```
help> modules
```

aquí mostrará los diferentes comandos que puede utilizar, para obtener mayor ayuda sobre determinado comando debe ejecutar lo siguiente:

```
help> os
```

siendo “os” un comando ejemplo. (Covantec R.L., 2018)

2.1.3.3. LIBRERÍA CUÁNTICA DE MICROSOFT ‘Q-SHARP’

Actualmente no se desarrollan códigos completamente cuánticos, en cambio se realizan una especie de combinaciones con computación clásica para generar códigos híbridos aplicando la mecánica cuántica en los comandos de operaciones más importantes del código.

El kit de desarrollo de Microsoft Quantum muestra muchas formas de aprender a desarrollar un programa cuántico. Desde la perspectiva técnica un programa cuántico es un conjunto de subrutinas clásicas que realizan determinadas operaciones en un sistema o máquina cuántica. Por diseño, Q# no define los estados cuánticos u otras propiedades de la mecánica cuántica directamente, sino que modifica los estados de estos qubits para lograr las operaciones necesarias.

La inicialización de un estado cuántico se hace creando un objeto de tipo $Qubit()$ y se aplica una puerta Hadamard con el fin de preparar el qubit para modificarlo en su estado de superposición, sí al qubit no se le aplica la puerta Hadamard, este se encontrará en estado fundamental y no se aprovecharía la superposición cuántica quien hace la diferencia con la computación clásica. Este proceso se muestra en la figura 7.

Q# Copiar

```
using (qubit = Qubit()) {
    // At this point, the qubit is in the state |0⟩.
    H(qubit);
    // H is now applied, such that the qubit is in H|0⟩ = |+⟩, as desired.
}
```

Fig. 7. Inicialización de un qubit. (Microsoft, 2020)

En el código anterior no se muestra explícitamente el estado cuántico, sino que describe la transformación del estado mediante el programa. Así, un programa no puede describir el estado, sino que realiza las modificaciones necesarias para que al momento de realizar la medición con ayuda de la operación “Measure” se obtenga la información esperada del qubit. Para realizar estas modificaciones se hace uso de las puertas cuánticas mencionadas en el numeral de computación cuántica. Las modificaciones que se realizan al qubit mediante las puertas cuánticas genera determinado resultado al ejecutarlo en la computadora cuántica, aunque este resultado es relativamente constante, también depende de la interpretación que le haga la computadora cuántica, es decir, en computadoras cuánticas reales el resultado será relativamente diferente a los resultados obtenidos mediante la ejecución del código con ayuda de un simulador cuántico.

“Un programa Q# vuelve a combinar estas operaciones, tal y como se define en un equipo de destino para crear nuevas operaciones de nivel superior para expresar el cálculo cuántico. De esta manera, Q# hace que sea fácil expresar el Quantum

subyacente de la lógica y los algoritmos de Quantum híbridos, mientras quetambién son generales con respecto a la estructura de un equipo de destino o un simulador.” Referencia textual de: (Microsoft, 2020).

En la sintaxis de un lenguaje se describen distintas combinaciones de símbolos que hacen que el programa se forme correctamente. Dentro de Q# estos elementos se dividen en tres grupos:

TIPOS: Este es un lenguaje bastante tipado, de modo que el correcto uso de los tipos ayuda a garantizar la correcta ejecución del código. Además de la compatibilidad de Q# con los tipos de datos estándar también existe compatibilidad con tipos definidos por el usuario. Para información detallada sobre los tipos de datos manejados o compatibles con la librería Q# se puede dirigir a la referencia: (Microsoft, 2020).

EXPRESIONES: Se refiere a la combinación de dos o más constantes, variables, operadores y funciones que el lenguaje interpreta y evalúa como un valor especifico. También pueden ser llamadas literales, “Por ejemplo, 5 es un Int literal (por lo tanto, una expresión de tipo Int) y si el símbolo $count$ se enlaza al valor entero 5, entonces $count$ también es una expresión de tipo entero.

INSTRUCCIONES: es una unidad sintáctica que expresa alguna acción que se llevará a cabo. Se diferencian con las expresiones en que estas son evaluadas mientras que las instrucciones se ejecutan.

Una de las principales instrucciones es la for, la cual admite iteración e incluye un bloqueo de instrucciones, un ejemplo de esta instrucción se muestra en la figura 8.

Q# Copiar

```
for (qubit in qubits) {
    H(qubit);
}
```

Fig. 8. Instrucción for. (Microsoft, 2020)

Este código, corre en iteración para cada qubit del registro o del circuito cuántico y realiza una Hadamard a cada uno de ellos. Además $H(qubit);$ también es una instrucción por sí misma.

2.1.3.4. LIBRERÍA CUÁNTICA DE IBM 'QISKIT'

Qiskit es un kit de desarrollo de software SDK abierto para trabajar en computadoras cuánticas a nivel de pulsos, circuitos y algoritmos con el uso de las puertas cuánticas. Acelera el desarrollo de aplicaciones cuánticas al proporcionar el conjunto completo de herramientas necesarias para la interacción con computadores y simuladores cuánticos.

Con esta herramienta se puede tener acceso a un amplio conjunto de circuitos generados por programadores especializados que pueden ser usados como base para bloques de construcción de circuitos cuánticos más complejos.

Además, se puede hacer uso de una biblioteca de algoritmos cuánticos para la investigación y desarrollo de aplicaciones de aprendizaje automático, optimización y química. Incluso se puede estudiar y reducir el impacto del ruido con ayuda de módulos para la caracterización del ruido y la optimización del circuito.

Los códigos generados con las características mencionadas además se pueden ejecutar en múltiples arquitecturas de hardware cuántico disponible en las máquinas de IBM.

Esta herramienta se pude utilizar en línea con el simulador de IBM o se pueden instalar las librerías dentro de un software como Python, Jupyter, entre otros.

El algoritmo general de desarrollo de un programa cuántico haciendo uso de Qiskit consta de tres pasos principales: construir, ejecutar y analizar. Para mostrar de forma general el paso de la producción de un código con Qiskit se realiza el siguiente flujo de trabajo:

i) Importar paquetes: Para importar los paquetes necesarios para el programa se digitan códigos como los mostrados en la figura 9.

```
import numpy as np
from qiskit import(
  QuantumCircuit,
  execute,
  Aer)
from qiskit.visualization import plot_histogram
```

Fig. 9. Código para importar librerías en Python. (Equipo de desarrollo de Qiskit , 2020)

ii) Se inicializan las variables y el circuito cuántico. Como se muestra en la figura 10, se inicializa un circuito cuántico con 2 qubits en el estado cero; dos bits clásicos inicializados en cero y $circuit$ es el circuito cuántico.

```
circuit = QuantumCircuit(2, 2)
```

Fig. 10. Inicialización del circuito cuántico. (Equipo de desarrollo de Qiskit , 2020)

iii) Se agregan las puertas necesarias u operaciones para manipular los registros del circuito cuántico. Teniendo en cuenta las líneas de código de la figura 11,

```
circuit.h(0)
circuit.cx(0, 1)
circuit.measure([0,1], [0,1])
```

Fig. 11. Aplicación de puertas cuánticas, Hadamard, CX y medición respectivamente. (Equipo de desarrollo de Qiskit , 2020).

Donde la primera línea de código, aplica una puerta Hadamard a qubit del circuito en posición cero, la segunda línea aplica una puerta control X con el qubit en posición cero como control y el de posición uno como objetivo. Finalmente se aplica la operación de medición, quien almacena la información registrada en los qubits en los bits clásicos conservando las posiciones de los mismos.

iv) Para ver el circuito que se lleva diseñado basta con aplicar la instrucción mostrada en la figura 12, aquí también se muestra el circuito que se va desarrollando hasta este punto.

```
circuit.draw()
```

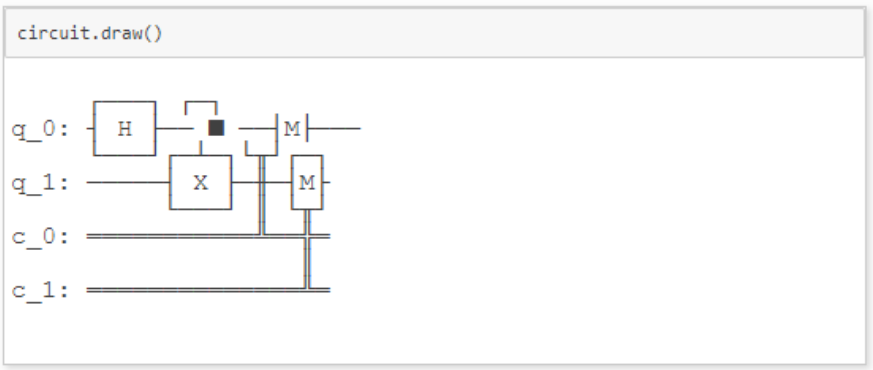

Fig. 12. Visualización del circuito cuántico. (Equipo de desarrollo de Qiskit , 2020).

El experimento se puede simular con ayuda de $Qiskit\ Aer$ o se puede usar el simulador $BasicAer$ el cual esta incluido en la librería Qiskit Terra. Para hacer uso

del simulador, se hace necesario realizar la importación del complemento con el código mostrados en la figura 13.

```
import numpy as np
from qiskit import(
  QuantumCircuit,
  execute,
  BasicAer)
...
```

Fig. 13. Importación de complementos de simulación. (Equipo de desarrollo de Qiskit , 2020).

v) Luego de tener importado el complemento se puede realizar la simulación haciendo uso de las líneas de código mostradas en la figura 14.

Fig. 14. Simulación del circuito cuántico. (Equipo de desarrollo de Qiskit , 2020).

```
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1000)
result = job.result()
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)
```

```
Total count for 00 and 11 are: {'11': 499, '00': 501}
```

vi) Para finalizar, se puede visualizar los resultados mediante un histograma para ello es indispensable importar la función $plot\_histogram$ (esta línea de importación se realizó en el paso i.). en la figura 15 se muestra el histograma que representa el resultado delcircuito ejemplo generado.

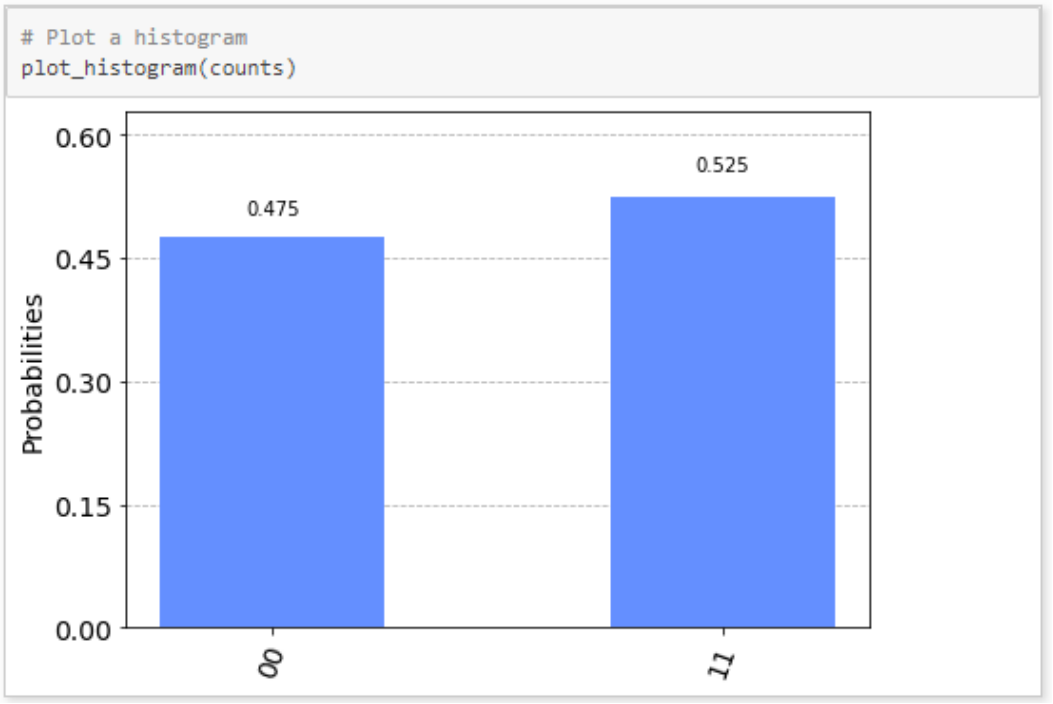

Fig. 15. Histograma del resultado del circuito. (Equipo de desarrollo de Qiskit , 2020).

Para mayor información sobre las librerías y complementos incluidos en el kit de desarrollo cuántico de IBM Qiskit, dirigirse a la referencia en cuestión: (Equipo de desarrollo de Qiskit , 2020).

3. ACTIVIDADES

Tabla 1. Cronograma de actividades manejado en la práctica.

	Mes	dic-19		ene-20				feb-20				mar-20				abr-20	
Objetivos	Actividades	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
Aprender y comprender las bases de inteligencia artificial y computación cuántica para ser aplicadas en un perceptrón cuántico.	Estudio de conceptos básicos de IA.																
	Estudio de conceptos básicos de computación y superposición cuánticas.																
	Investigación de últimos avances en IA y Machine Learning.																
Conocer y usar la programación mediante Python en computación clásica para generar un código híbrido entre programación clásica y cuántica.	Estudio de los principios básicos de Python																
	Instalación del software en el equipo personal.																
Repasar y aplicar las librerías Qiskit de IMB para la programación del perceptrón cuántico.	Instalación de las librerías cuánticas de IBM (Qiskit).																
	Estudio de las librerías Qiskit.																
Programar un perceptrón sencillo en computación cuántica.	Investigación sobre antecedentes de perceptrones en computación cuántica.																
	Programación del perceptrón cuántico simple y validación.																
Extra	Mejora del perceptrón simple agregando una capa oculta.																
	Diseño de Simpletrón.																

4. METODOLOGÍA

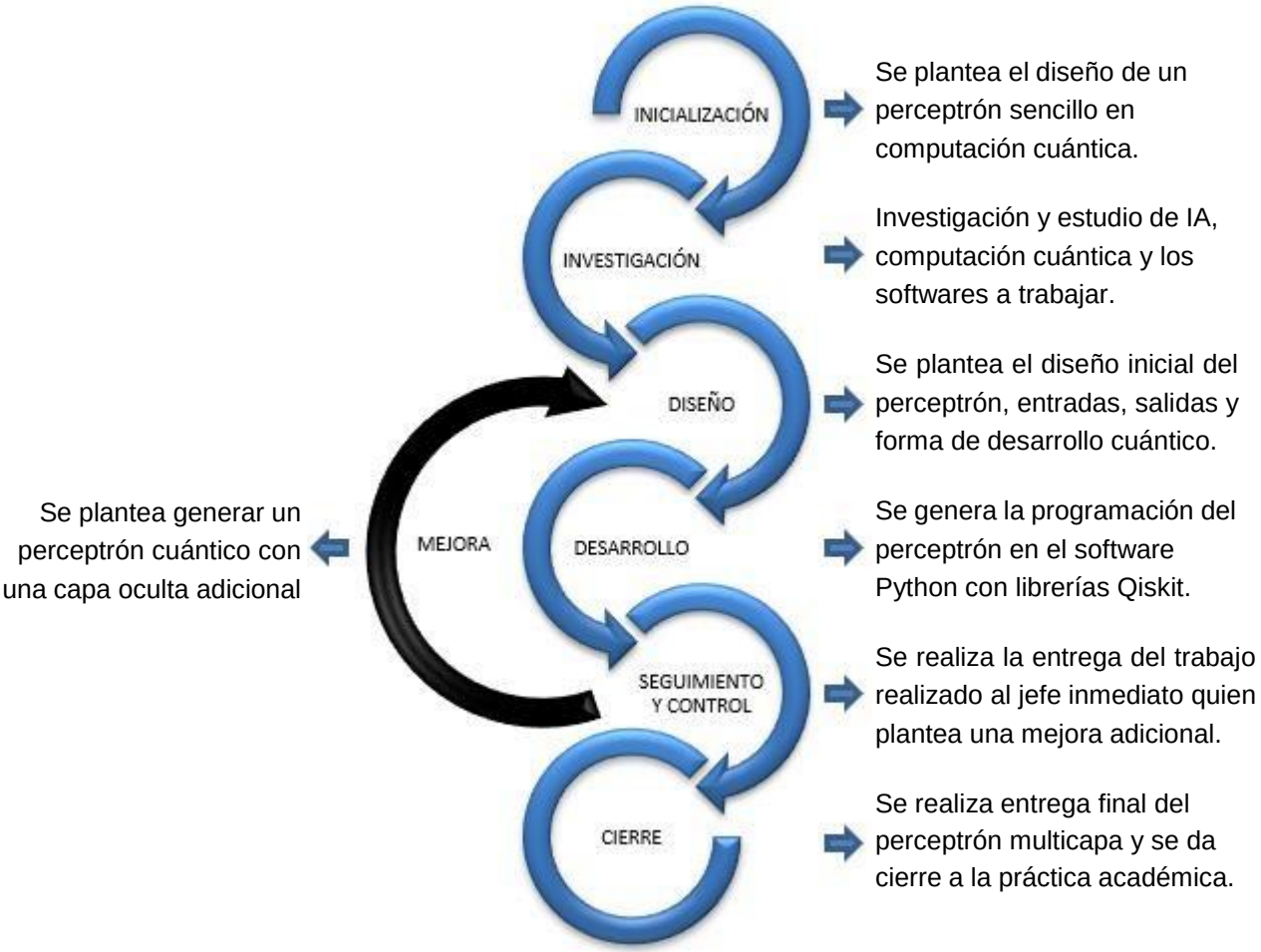

Fig. 16. Metodología en cascada. (Lozano, 2020).

La metodología que se trabajó fue la “Metodología de proyectos en cascada”. Para el desarrollo de la practica académica primero se plantearon los objetivos los cuales fueron presentados a la universidad y aprobados por el director de prácticas y el jefe inmediato en la empresa Grupo Bernier SAS.

El principal objetivo fue el diseño de un perceptrón sencillo en computación cuántica con aplicación de clasificación de dos entradas independientes en dos tipos de datos. Luego de plantear la meta de la práctica se procedió a realizar un estudio de los temas relacionados con el trabajo que se deseaba realizar durante las primeras semanas, además mientras se avanzaba en el trabajo se continuó

realizando actividades de investigación. Para el correcto desempeño dentro de la empresa se realizó investigación sobre las teorías básicas de computación cuántica e inteligencia artificial. Terminada la etapa de investigación se procedió a realizar el diseño del perceptrón que se deseaba implementar, donde inicialmente se decidió generar un perceptrón sencillo con dos neuronas de entrada y una neurona de salida de la forma mostrada en la figura 17.

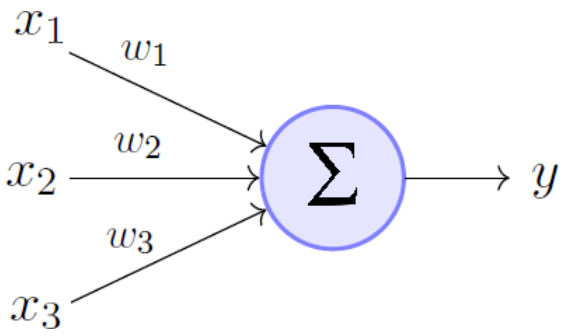

Fig. 17. Diseño general del perceptrón simple. (Torres Barrán & Naveiro, 2019)

Posteriormente, se implementó el perceptrón cuántico en Python con las librerías de IBM (Qiskit). Para la implementación del perceptrón se tuvo en cuenta el principio de distancia de los estados cuánticos, convirtiendo cada uno de los datos de entrada en un estado de superposición y comparando la distancia de dichos datos con respecto a los estados primarios $|1>$ y $|0>$.

Al finalizar la etapa de diseño se presentaron los resultados al jefe inmediato de forma virtual. Para la entrega tanto de los avances como del resultado final se usó la metodología virtual, donde teniendo el algoritmo diseñado o los avances en cuestión se realizaba un video explicativo para mostrar y especificar las acciones realizadas y la forma de abordar las actividades asignadas por el jefe inmediato. Es de gran importancia aclarar que además de las entregas de resultados se vino realizando un acompañamiento constante por parte de este, tanto en la etapa de investigación como en la de diseño y desarrollo del algoritmo. El señor Jean Bernier, recibió el algoritmo del perceptrón sencillo con la explicación del desarrollo de este de forma satisfactoria y propuso realizar un avance extra a los objetivos planteados inicialmente; dicho avance refiere a agregar una capa oculta al perceptrón cuántico diseñado.

Para realizar esta mejora, se tuvo que regresar a la etapa de diseño para incluir la nueva capa, así, el diseño del perceptrón quedó finalmente como se muestra en la figura 18.

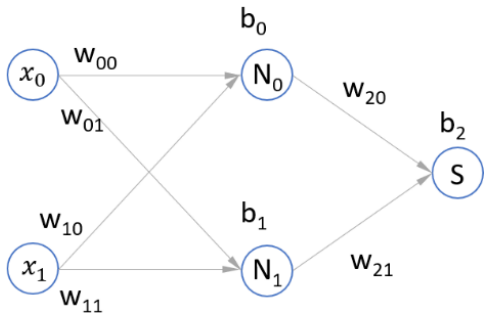

Fig. 18. Estructura general del perceptrón multicapa. (Burrueco, s.f.)

Finalmente se implementó el perceptrón multicapa y se presentaron los resultados al jefe inmediato el cual aprobó la entrega y con esto se dio por terminada la práctica académica de forma satisfactoria el día 16 de abril de 2020.

5. RESULTADOS

1. Estudio de conceptos básicos de IA.

En las primeras semanas se propuso realizar estudio e investigación sobre inteligencia artificial, con el fin de tener fresco temas relacionados con redes neuronales, más específicamente el perceptrón sencillo y perceptrón multicapa, con implementación de aprendizaje Backpropagation. Dicha etapa de investigación y sus resultados juntos con las bases bibliográficas se encuentran en este documento dentro de la sección de marco teórico, el cual se evidenció anteriormente en la sección 2.1.1.

2. Estudio de conceptos básicos de computación cuántica y superposición cuántica.

Por otra parte, también se hizo necesario es estudio de computación cuántica, el principio de superposición y temas relacionados. Dicho estudio se realizó, mediante una primera etapa de investigación, y posteriormente se realizó una parte de familiarización con las librerías que componen el lenguaje cuántico dentro del amiente de IBM, aquí, se realizaron algunas prácticas con el simulador virtual

con el fin de observar el comportamiento de los qubits al aplicarles cada una de las compuertas lógico-cuánticas.

La información teórica obtenida gracias al proceso de investigación realizado en este numeral, se encuentra registrada en la sección de marco teórico junto con la información referente a las librerías y el software utilizado. Para revisar dicha información se puede dirigir al numeral 2.1.2.

3. Investigación de últimos avances en IA y Machine Learning

Como una de las tareas asignadas por el señor Jean Bernier, jefe inmediato en la práctica académica fue el realizar una investigación de los últimos avances sobre Deep Learning y Machine Learning. Actividad que se realizó satisfactoriamente y se obtuvo la información que se muestra en las tablas 2 y 3.

Tabla 2. Información sobre avances en Deep Learning.

DEEP LEARNING

	APROXIMACIONES	
TÍTULO	AI architecture has quantum computing aspirations. EENews Europe. Enero *08, 2020 //By Peter Clarke*	Alpha Zero Algorithm Applied to Quantum Computing. *Enero 23, 2020 // By Alex McFarland*
TECNOLOGÍA USADA	Quantum Flow AI SoC: • Procesador RISC-V clásico. • Generador QuantaFlow. • Espacio QF Evolution	Emparejamiento de Alpha Zerocon un algoritmo especializado de computación cuántica.
PAÍS DE ORIGEN	California (Fremont)	Dinamarca (Aarhus)
EMPRESA O ENTIDAD	PQ Labs Inc.	Grupo de investigación. Universidad de Aarhus
LINKEDIN	Correo: contact@pqlabs.ai	Christian Fischer Pedersen / Stefan Rahr Wagner, Departamento de Ingeniería Correo electrónico: cfp@eng.au.dk / sw@eng.au.dk / sherson@phys.au.dk Teléfono: +4541893318 / +4541893254

DESCRIPCIÓN	Esta arquitectura simula un espacio de transformación para registros de Qf-bits. Implementando un procesador RISC-V clásico de un solo núcleo de modo que realice el papel de controlador lógico, recuperación de la información física que se va a tratar. Los datos obtenidos en baja dimensión son transformados en alta dimensión gracias al generador QuantaFlow para iniciar la transformación y evolución continua de la información. Para terminar el proceso, la lectura de la información desde el espacio de evolución es realizada por medio de la unidad Bit Observer. Además, si es necesariauna deformación mayor para el espacio de evolución, el procesador ejecuta un reinicio en caliente del espacio deevolución.	El grupo de investigación realizó simulaciones por computadora para demostrar cómo AlphaZero puede ser aplicado a tres tipos de problemas de control. Según el estudio realizado: "AlphaZero emplea una red neuronal profunda junto con una búsqueda anticipada profunda en una búsqueda de árbol guiada, que permite la aproximación predictiva de variables ocultas del paisaje de parámetros cuánticos. Para enfatizar la transferibilidad, aplicamos y comparamos el algoritmo en tres clases de problemas de control usando solo un conjunto común de hiper parámetros algorítmicos ". NOTA: La investigación realizada por el equipo fue publicada en *Nature Quantum Information.*
ENLACE DE REFERENCIA	https://www.prnewswire.com/news-releases/ces-2020-ai--quantum-flow-boosts-deep-learning-speed-10x---15x-faster---powered-by-pqlabsai-300983793.html	Documento informativo: https://www.unite.ai/alphazero-algorithm-applied-to-quantum-computing/ Investigación completa: https://www.nature.com/articles/s41534-019-0241-0

RESULTADOS LOGRADOS	- Con QuantaFlow es posible ejecutar todo tipo de modelos de redes neuronales, por ejemplo, ResNet-50 (2015), MobileNet (2017), EfficientNet (2019), etc.) sin degradar la velocidad o golpear el "muro de memoria". - Hot-Patching se puede utilizar para cambiar la ruta de evolución de qf-bits dinámicamente.	- El algoritmo aprendió a explotar una simetría subyacente del problema que originalmente no fue considerada. - AlphaZero es capaz de aprender por sí solo sin ninguna interferencia de los humanos.
APLICACIONES ACTUALES	• Industrias y fabricación • Transporte • Fintech • Medios y radiodifusión • Viajes/Hospitalidad • Comercio minorista y electrónico • Salud y medicina • Seguridad y vigilancia	- Se encuentra en proceso de experimentación - Autoaprendizaje - Videojuegos
ALGORITMO IMPLEMENTADO	Red neuronal con arquitectura ResNet-50	Nota: *"Todo el código utilizadopara generar los datos presentados en este documento está disponible a pedido. Todas lassolicitudes deben dirigirse a J. Sherson (sherson@phys.au.dk)."*

Tabla 3. Información sobre avances en Deep Learning parte 2.

	Aproximaciones
TÍTULO	First chip-to-chip quantum teleportation harnessing silicon photonic chip fabrication *Comunicado de prensa: 23 December 2019*
TECNOLOGÍA USADA	Chips que aprovechan la generación y manipulación de partículas de luz individuales dentro de circuitos programables de nano escala. "Este experimento fue posible gracias a la tecnología de fotónica de silicio de baja pérdida de vanguardia basada en la fabricación de alta calidad en la DTU"
PAÍS DE ORIGEN	Reino Unido (Brístol) Dinamarca (Kongens Lyngby)
EMPRESA O ENTIDAD	University of Bristol Technical University of Denmark
LINKEDIN	Correo: dan.llewellyn@bristol.ac.uk imad.faruque@bristol.ac.uk Tel. +44 (0) 117 33 15373
DESCRIPCIÓN	Cada chip fue programado completamente con el fin de obtener diferentes demostraciones que utilizan el enredo de qubits. “La demostración principal fue un experimento de teletransportación de dos chips, mediante el cual el estado cuántico individual de una partícula se transmite a través de los dos chips después de realizar una medición cuántica. Esta medición utiliza el extraño comportamiento de la física cuántica, que simultáneamente colapsa el enlace de enredo y transfiere el estado de la partícula a otra partícula queya está en el chip receptor ". Todas las fuentes son probadas y obtienen que son casi idénticas y emiten fotones con el mismo parentesco. Estos chips pueden codificar información cuántica en forma de la luz generada dentro de los circuitos y pueden procesar dicha información con alta eficiencia y baja cantidad de ruido.

LINK DE REFERENCIA	Artículo de revista: https://www.bristol.ac.uk/news/2019/december/quantum-teleportation.html Investigación completa: https://www.nature.com/articles/s41567-019-0727-x
RESULTADOS LOGRADOS	• Demostración de la teletransportación cuántica de información entre dos chips programables. • Teletransportación cuántica de extremadamente alta fidelidad del 91 por ciento. • Con base en las fuentes de fotones individuales de alta calidad en chip, fue construido un circuito aún más complejo que contiene cuatro fuentes.
APLICACIONES ACTUALES	- Intercambio de enredos (requerido para repetidores cuánticos y redes cuánticas) - Estados GHZ de cuatro fotones (requeridos en computación cuántica e internet cuántica).
ALGORITMO	Nota: El código de computadora utilizado para el análisis de datos está disponible a solicitud del autor correspondiente.

MACHINE LEARNING

Tabla 4. Tipos de algoritmos de Machine Learning.

TIPO DE ALGORITMO	RESUMEN
ALGORITMOS BASADOS EN INSTANCIAS	Estos métodos de aprendizaje guardan los datos de entrenamiento de modo que cada vez que una nueva instancia es encontrada se realiza el cálculo de la relación entre dicha instancia y los datos guardados, de modo que se genera un valor de la función objetivo para la instancia nueva. Este método incluye al vecino más cercano y a determinados métodos de regresión que hacen que las instancias sean representadas como punto de un espacio euclídeo. También incluye métodos de razonamiento dependiente de casos. Es un método de aprendizaje lento ya que se genera o estima la función objetivo para cada una de las instancias que intervienen.

ALGORITMOS DE REDUCCIÓN DE DIMENSIÓN	Estos algoritmos mapean el conjunto de datos a subespacios generados a partir del espacio original con dimensiones más pequeñas de modo que se optimiza el costo en el procesamiento de los datos. Aprovecha la estructura nosupervisada existente para poder realizar la compresión de los datos, además, facilita la representación gráfica de los modelos más complejos o con múltiples características en su presentación original.
ALGORITMOS DE ÁRBOL DE DECISIÓN	Refiere a un árbol en el que cada uno de los nodos de rama representa una selección entre diferentes alternativas y cada nodo de hoja una posible decisión. Es un método de aprendizaje supervisado usado principalmente en selección y clasificación y funciona para entradas y salidas de tipo categóricas y continuas. Es uno de los más utilizados en el proceso de llegar a una conclusión a partir ejemplos determinados y en la minería de datos. Nota: “Los árboles de decisión aprenden y se entrenan a partir de ejemplos dados y predicen para circunstancias no vistas.” Se usa principalmente en casos en los que, se puede describir los ejemplos en términos de atributo-valor, la función objetivo toma valores discreto, existe posibilidad de ruido en el conjunto de entrenamiento y cuando no se conocen todos los valores de los atributos de los ejemplos.
ALGORITMOS DE REGRESIÓN	Se trata de un algoritmo de aprendizaje supervisado que normalmente se utiliza en estadística y machine Learning. La versión más elemental de este método consiste en trazar una curva que representa la tendencia de los datos, sean continuos o discretos. En general se refiere a una aproximación de una variable dependiente ‘Y’ de una o más variables de entrada ‘X’. Teniendo en cuenta que el objetivo general del machine Learning es aprender automáticamente, entonces, en este caso el algoritmo genera por sí mismo la curva de tendencia que simula el comportamiento del sistema que se desea predecir. Para ello, mide los valores tanto de entrada como de salida paraobtener el valor de error o desfase, con este valor de modifica elmodelo de la curva generada, para así, ir disminuyendo el valor de error o coste del sistema.

ALGORITMOS DE CLUSTERING O DE AGRUPACIÓN	Consiste en agrupar una serie de vectores según un criterio en grupos o Clúster. Generalmente el criterio suele ser la similitud por lo que se puede decir que agrupa vectores similares en grupos. En general, para realizar el proceso de agrupación primero que todo se define el número de grupos o clusters en los cuales se va a distribuir los datos o vectores, como segunda medida se forman grupos con similitudes y se asigna un núcleo o centro a cada conjunto. Tercero, se mide el error en la capacidad que tiene la máquina para encontrar cada uno de los centros de los grupos generados, luego se vuelven a generar nuevos grupos aplicando un factor de rectificación partiendo del centro obtenido anteriormente. Finalmente se mide el error y se agrega al algoritmo hasta que este ya no varíe.
ALGORITMOS DE REDES NEURONALES	Las redes neuronales imitan el comportamiento de activación e interconexión entre redes neuronas biológicas con el fin de encontrar soluciones a problemas complejos. Se suelen utilizar para problemas de clasificación y regresión, pero realmente tienen un gran potencial para resolver multitud de problemáticas. Son muy buenas para detectar patrones. Las Redes Neuronales Artificiales requieren mucha capacidad de procesamiento y memoria y estuvieron muy limitadas, sin embargo, dieron labase para la creación de los algoritmos de aprendizaje profundo.
ALGORITMOS DE APRENDIZAJE PROFUNDO O DEEP LEARNING	Este método de aprendizaje es un aspecto de la inteligencia artificial que imita el aprendizaje que los seres humanos utilizamos para obtener determinados conocimientos. Es una rama del aprendizaje automático. Sin embargo, los sistemas de aprendizaje profundo a diferencia de los tradicionales, mejora su rendimiento al poder acceder a una cantidad de datos mayor, es decir, la máquina obtiene mayor experiencia. Estos algoritmos aprenden al crear modelos computacionales compuestos por varias capas de procesamiento y detención de estructuras más complejas de datos.

ALGORITMOS BAYESIANOS	Refiere al proceso de encontrar la hipótesis más probable de acuerdo con un conjunto de ejemplos de entrenamiento y un conocimiento de la probabilidad de cada hipótesis. Se caracteriza porque cada ejemplo de entrenamiento afecta la probabilidad de las hipótesis, se puede incluir conocimiento de la probabilidad de cada hipótesis, se asocia un porcentaje de confianza a las predicciones y combina predicciones en base a su confianza. Además, una nueva instancia es clasificada como función de las predicciones de diferentes hipótesis de acuerdo con las probabilidades de cada una. Incluso, en casos en sea imposible dar solución al problema mediante este método, se obtiene una aproximación que lleve a una posible solución.
PROCESAMIENTO DEL LENGUAJE NATURAL g(NLP)	Es una rama de la inteligencia artificial que ayuda a las computadoras a entender, interpretar y manipular el lenguaje humano. Este algoritmo toma elementos prestados de muchas disciplinas, incluyendo la ciencia de la computación y la lingüística computacional, con el fin acercar la comunicación humana y el entendimiento de las computadoras. El procesamiento del lenguaje natural incluye diferentes técnicas para interpretar el lenguaje humano, entre ellos seencuentran los métodos estadísticos, el aprendizaje basado en máquina o incluso estrategias basadas en reglas y algoritmos. “Las tareas básicas de NLP incluyen la simbolización y el análisis sintáctico, lematización/derivación, etiquetado de la parte del habla, detección del lenguaje e identificación de relaciones semánticas. Si alguna vez creó diagramas de enunciados en la primaria, ya ha realizado estas tareas de forma manual antes.”

4. Estudio de los principios de básicos de Python y librerías cuánticas de IBM Qiskit.

Se realizó el estudio detallado de la programación en Python tanto aplicando las librerías cuánticas de IBM, como la programación clásica con lenguaje C++, para generar el código híbrido del perceptrón cuántico. La información general sobre Python y las librerías utilizadas se encuentra mencionada en el numeral de marco conceptual. En dicha sección, se muestra la forma general de programación en el software además de una parte teórica del mismo, esta información se encuentra en el apartado 2.1.3.2 y 2.1.3.4.

5. Instalación del software en el equipo personal y librerías cuánticas de IBM (Qiskit).

La instalación del software necesario se realizó con ayuda de la plataforma anaconda, donde inicialmente se instaló Jupyter ya que Python no permitía la aplicación de las librerías de Qiskit de IBM, se inició el proceso de estudio y validación del funcionamiento de cada una de las puertas cuánticas con ayuda de este software, sin embargo, se realizó el proceso de desinstalación total delPython con el fin de desarrollar este proceso de nuevo desde cero. Es importante aclarar que, aunque inicialmente se trabajó sobre Jupyter, el lenguaje de programación utilizado fue el de Python. Al realizar dicha acción se logró instalar correctamente tanto el software Python, como las librerías de computación cuántica de IBM. En este punto se pudo iniciar el proceso de digitalización del código del perceptrón cuántico.

6. Investigación sobre antecedentes de perceptrones en computación cuántica.

Para el desarrollo del diseño y digitalización del algoritmo de inteligencia artificial mencionado en este documento, se hizo necesaria una etapa de investigación sobre trabajos similares. Dicha investigación arrojó la información mencionada en la tabla 5, donde se plasma los principios básicos de algunos algoritmos deperceptrón.

Tabla 5. Algoritmos de perceptrón cuántico similares. (Bon, 2018;2019), (Schuld & Petruccione, 2014)

QUANTUM ALGORITHM FOR THE IMPLEMENTATION OF A PERCEPTRON.	SIMULATING A PERCEPTRON ON A QUANTUM COMPUTER
Implementación de un perceptrón con una sola neurona y con vectores de entrada y pesos de valor 1. Primero se tienen en cuenta las contrapartes cuánticas de dichos vectores con: $\theta = J_1\|\psi\rangle = \frac{1}{\sqrt{(2^N)}} \sum_{j=0}^{2^N-1} i_j\|j\rangle$ $\|\psi_w\rangle = \frac{1}{\sqrt{2^N}} \sum_{j=0}^{2^N-1} i_w\|j\rangle$ $i_j, w_j \in \{-1,1\}$ Luego, para poder hacer el productodel vector de entrada con el vector de peso se tiene en cuenta que, $\langle\psi_i\| U^{\dagger}U \|\psi_w\rangle = \langle\psi_i\|\psi_w\rangle$ El producto interno en el lado derecho de la última fórmula se obtiene utilizando un qubit ansilla y una puerta CNOT con todos los qubits sobre los que los vectores de entrada y peso actúan, trabajando como qubits controlados, y la ansilla trabajandocomo objetivo. Gracias a esto, al medir el qubit ansilla se obtiene la salida que puede ser un perceptrón activado o desactiva, es decir, 1 o 0.	Está basado en la idea de normalizar el vector de entrada en la fase del estado cuántico y aplicando el algoritmo de estimación de fase. Esta es la representación de la fracción binaria de $\theta = J_1 * \frac{1}{2} + \cdots + J_2 \frac{1}{2^t}$, lo cual indica que θ es mayor que $\frac{1}{2}$. Iniciando con la aplicación de puertas Hadamard con el objetivo de colocar los estados inicializados en |0> en estado de superposición para posteriormente aplicar las puertas U quienes normalizan las entradas binarias a estados cuánticos. Se aplica una transformada inversa de Fourier con el fin de analizar los estados y obtener la salida del perceptrón, como se muestra en la siguiente ecuación. $\frac{1}{\sqrt{(2^N)}} \sum_{j=0}^{2^T-1} \exp^{2\pi i j\varphi} \|\psi_w\rangle \xrightarrow{QFT^{-1}} \sum_{j=0}^{2^T-1} \left(\frac{1}{2^T} \sum_{k=0}^{2^T-1} \exp^{2\pi i k(\varphi - \frac{j}{2^T})}\right) \|J\rangle$ Para la representación cuántica del vector de entrada, se debe reemplazar la operación de parametrización con la matriz mostrada. $U_{W_k^{(m)}, x_k} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-2\pi\delta\phi\frac{1}{2^m}} & 0 \\ 0 & 0 & 0 & e^{-2\pi\delta\phi\frac{1}{2^m}} \end{pmatrix}$

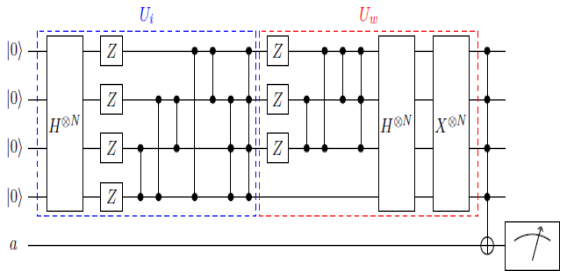

Fig. 19. Circuito cuántico 1

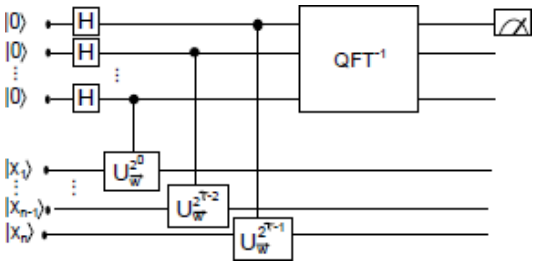

Fig. 20. Circuito cuántico 2

Tabla 6. Algoritmos de perceptrón cuántico similares 2. (Schuld, Fingerhunth, & Petruccion, Quantum Reseach Group, School of chemistry and physics, 2017)

IMPLEMENTING A DISTANCE-BASED CLASSI_ER WITH A QUANTUM INTERFERENCE CIRCUIT
El circuito implementado utiliza dos vectores de entrenamiento y un vector de entrada escalado y normalizado a partir de dos entradas de coordenadas que se relacionan entre sí para obtener una salida con dos únicas respuestas 1 o 0. Primero se colocan los qubits de ansilla e índice en superposición, el vector de entrada es unido con el estado base de la ansilla. Entonces el vector de entrenamiento X_0 es relacionado con el estado excitado de la ansilla y el estado base del qubit índice, seguido por la unión entre el vector de entrenamiento X_1 con el estado excitado de la ansilla y el qubit índice. Luego, los qubits dato y clase son intercambiados y el qubit clase es rotado de acuerdo con el qubit índice siendo $\|1\rangle$, lo cual completa la preparación del estado inicial. Finalmente se aplica una puerta Hadamard que interfiere los números del vector de entrada con los vectores de entrenamiento y se mide la ansilla seguido de una medición al qubit clase cuando se encuentra que el qubit ansilla está en el estado $\|0\rangle$.

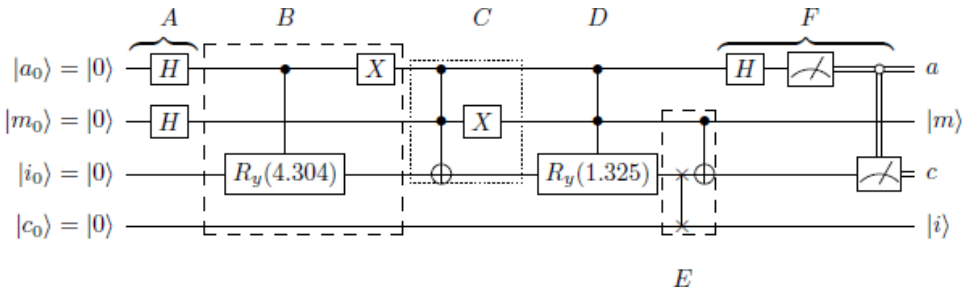

Fig. 21. Circuito cuántico implementado.

7. Programación del perceptrón cuántico simple y validación.

La programación del perceptrón cuántico simple se realizó en el software Python con uso de las librerías de del kit de IBM Qiskit. Para la codificación del algoritmo cuántico se siguió el siguiente procedimiento:

I. Se realizó la importación de las librerías necesarias para la ejecución del código.

```
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute, BasicAer
from matplotlib import pyplot as plt
from qiskit.visualization import plot_histogram
import math
```

Fig. 22. Importación de librerías. (Lozano, 2020)

II. En la ejecución general se parte por solicitar al usuario los vectores de entrenamiento, iniciando con el valor de entrenamiento para la clase 0 y posteriormente el vector de entrenamiento para la clase 1.

```
print("              ___________ PERCEPTRÒN CUÀNTICO ___________")
while(n == 0):
    if (n1 == 0):
        print("                ***  ** DATOS DE ENTRENAMIENTO ** ***")
        print("Para iniciar, por favor ingrese las coordenadas del punto núcleo de la clase 0: ")
        x1 = float(input('x: '))
        y1 = float(input('y: '))
        print("Ahora, por favor ingrese las coordenadas del punto núcleo de la clase 1: ")
        x2 = float(input('x: '))
        y2 = float(input('y: '))
```

Fig. 23. Inicialización de los vectores de entrenamiento. (Lozano, 2020)

III. Se solicitan las coordenadas del valor a clasificar o vector de entradas, dependientes de dos variables físicas x e y.

```
print("Por favor ingrese las coordenadas del punto que desea clasificar: ")
x = float(input('x: '))
y = float(input('y: '))
x_prime = [x, y] # x' = DATO DE ENTRADA A CLASIFICAR
```

Fig. 24. Inicialización del vector de entradas. (Lozano, 2020)

IV. Se guardan los datos de los vectores de entrenamiento en un solo vector para ser utilizado más adelante.

```
    training_set = [
        ([x1, y1], 0), # COORDENADA DEL DATO DE ENTRENAMIENTO A CLASIFICACIÓN 0
        ([x2, y2], 1)] # COORDENADA DE DATO DE ENTRENAMIENTO A CLASIFICACION 1
    print(f"Clasificando x' = {x_prime} con ruido 'Simulator backend'")
```

Fig. 25. Almacenamiento de los vectores de entrenamiento como un solo vector. (Lozano, 2020)

V. Se procede a llamar la primera función la cual hace uso tanto del vector de entrenamiento como del vector de entradas.

```
class_result = classify(test_vector=x_prime, training_set=training_set)
print(f"El vector de entrada x' es de la clase {class_result}\n")
```

Fig. 26. Llamado de función general de clasificación. (Lozano, 2020)

VI. Dicha función $classify()$, hace uso de la función $initialize\_registers()$ la cual inicializa los registros del circuito cuántico, tomando 4 como el número de registros ya que se utilizan el qubit de ansilla, el qubit de índice, el qubit de dato y el de clase. Este número define el número de qubits y bits clásicos que intervienen en el circuito cuántico.

```
def initialize_registers(num_registers):
    q = QuantumRegister(num_registers)
    c = ClassicalRegister(num_registers)
    # Asignacion de nombre a los qubits
    ancilla_qubit = q[0]
    index_qubit = q[1]
    data_qubit = q[2]
    class_qubit = q[3]
    return (q, c, ancilla_qubit, index_qubit, data_qubit, class_qubit)
```

Fig. 27. Función de inicialización de registros. (Lozano, 2020)

VII. Se llama a la función $get\_angles()$ para normalizar el vector de entradas dependiente de valores x, y por la representación de los mismos en coordenadas polares. Esta normalización se hace de forma similar para todos los vectores de entrenamiento y entrada.

```
#__________ VECTOR DE ENTRADA _________________#
    x = test_vector[0]
    y = test_vector[1]
    if(x<0 and y<0):
        alpha = math.pi + abs(math.atan(y/x))
    elif (x<0 and y>0):
        alpha = math.pi - abs(math.atan(y/x))
    elif (x>0 and y<0):
        alpha = 2*math.pi - abs(math.atan(y/x))
    elif (x==0):
        if (y>0):
            alpha = math.pi/2
        else:
            alpha = 3*math.pi/2
    elif (y==0):
        if (x>0):
            alpha = 0
        else:
            alpha = math.pi
    else:
        alpha =  abs(math.atan(y/x))
    if (x==0 and y==0):
        print("La coordenada (0,0) no puede ser clasificada")
    else:
        angles.append(alpha)
```

Fig. 28. Función de normalización de los vectores de entrada y entrenamiento. (Lozano, 2020)

VIII. Luego se llama a la función $create\_circuit()$, por medio de la cual se genera el circuito cuántico necesario para realizar la clasificación del vector de entrada.

Dentro de este circuito cuántico, inicialmente se aplican puertas Hadamard a los qubit de ansilla e índice para colocarlos en

superposición, luego se aplica una puerta CNOT al qubit de dato con la ansilla como control y se realiza la rotación necesaria al qubit de dato para posicionarlo en la coordenada correspondiente de entrada. Finalmente se voltea el qubit ansilla para mover el vector de entrada al estado cero de la ansilla

Si el vector a posicionar en la esfera cuántica es uno de los vectores de entrenamiento, entonces se aplica la CNOT al qubit de dato con el índice como control y se realiza la rotación de la misma forma que para el vector de entrada. Luego, para el primer vector de entrenamiento se aplica la puerta X a qubit índice y si es el segundo vector de entrenamiento se aplica al qubit clase, con el fin de voltear a cada unoen su caso y así mover el vector de entrenamiento al estado cero de cada qubit.

Finalmente se aplica una puerta Hadamard para completar el circuito y poder aplicar la puerta de medición para proceder con la simulación y análisis del resultado obtenido.

El código referente a esta función se muestra en el anexo 1.

IX. Se llama la función $simulate()$, la cual inicializa la simulación del circuito cuántico generado en el paso anterior por medio de el simulador cuántico $'qasm\_simulator'$

```
simulate(quantum_circuit):
# Simulación de ruido
backend_sim = BasicAer.get_backend('qasm_simulator')
job_sim = execute(quantum_circuit, backend_sim)
result = job_sim.result()
counts = result.get_counts(quantum_circuit)
 plot_histogram(counts)
 plt.show()
return job_sim.result()
```

Fig. 29. Función de simulación del circuito cuántico. (Lozano, 2020)

X. Se realiza la interpretación de los resultados por medio de la función $interpret\_results()$ donde se analiza si los resultados de la simulación tienden a clase 0 o 1 de acuerdo con la probabilidad que tiene cada clase. Definiendo la función lambda quien recupera solo los resultados donde el qubit ansilla está en clase $|0\rangle$

```python
def interpret_results(result_counts):

    total_samples = sum(result_counts.values())
    # Define la función lambda que recupera solo resultados donde el qubit ancilla está en el estado |0>
    post_select = lambda counts: [(state, occurences) for state, occurences in counts.items() if state[-1] == '0']
    # Realiza la postselección
    postselection = dict(post_select(result_counts))
    postselected_samples = sum(postselection.values())
##      print(f'Ancilla post-selection probability was found to be {postselected_samples/total_samples}')
    retrieve_class = lambda binary_class: [occurences for state,
                                           occurences in postselection.items() if state[0] == str(binary_class)]
    prob_class0 = sum(retrieve_class(0))/postselected_samples
    prob_class1 = sum(retrieve_class(1))/postselected_samples
    print(f'La probabilidad para clase 0 es de: {prob_class0}')
    print(f'La probabilidad para clase 1 es de: {prob_class1}')

    return prob_class0, prob_class1
''' ---------------------------------------------------------------------------------------------------------
```

Fig. 30. Función de interpretación de los resultados obtenidos en la simulación. (Lozano, 2020)

XI. Finalmente, como el valor resultante es un porcentaje se realiza la clasificación con ayuda de la probabilidad mayor, es decir, si es más probable que el vector de entrada pertenezca a clase 1 entonces se tomará así, en caso contrario será de clase 0.

```python
prob_class0, prob_class1 = interpret_results(result.get_counts(qc))
if prob_class0 > prob_class1:
    return 0
elif prob_class0 < prob_class1:
    return 1
else:
    return 'Inconcluso. Resultados: 50/50'
```

Fig. 31. Finalización de la función de clasificación $classify()$. (Lozano, 2020)

XII. En este punto se culmina la ejecución de la función $classify()$ y se procede a preguntar al usuario si desea clasificar otro punto. De ser así se repiten el algoritmo desde el numeral III y cuando el usuario no desea ingresar más vectores de entrada se muestra la gráfica de los puntos clasificados; señalando los de clase 0 con color celeste y los de clase 1 con color morado.

```
    if (class_result == 0):
        Xo.append(x)
        Yo.append(y)
    else:
        Xl.append(x)
        Yl.append(y)
    continuar = input('Para clasificar otro punto digite
    if (continuar == 'Y' or continuar == 'y'):
        nl = 1;
    elif (continuar == 'R' or continuar == 'r'):
        nl = 0;
    else:
        n = 1;
plt.plot(Xo,Yo,'o',color= 'c')
plt.plot(Xl,Yl,'o',color= 'm')
plt.plot(x1,y1,'o',color= 'b')
plt.plot(x2,y2,'o',color= 'r')
plt.axis([-10,10,-10,10])
plt.show()
```

Fig. 32. Finalización de la ejecución general del perceptrón y dibujo de los puntos clasificados en el plano x, y. (Lozano, 2020)

El resultado obtenido mediante este código de perceptrón sencillo se muestra en la figura 33.

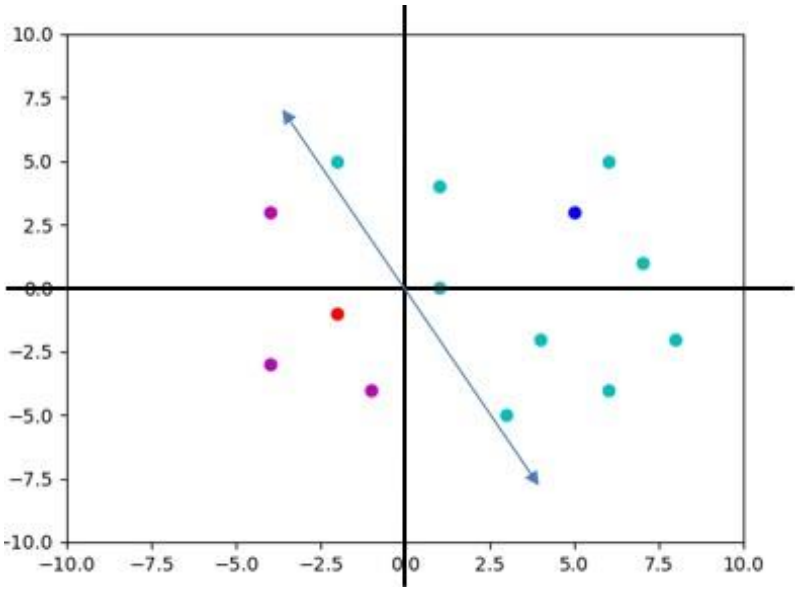

Fig. 33. Ejemplo de resultado final de clasificación mediante perceptrón cuántico sencillo. (Lozano, 2020)

8. Mejora del perceptrón simple agregando una capa oculta.

Luego de la entrega del perceptrón sencillo al jefe inmediato, este sugirió la mejora del mismo teniendo en cuenta que se había logrado el objetivo de la practica académica en menor tiempo de lo esperado, él mencionó que se podría generar un perceptrón con una capa extra para darle mayor potencia.

El perceptrón sencillo solo puede hacer clasificación en casos en los que se tienen únicamente dos vectores de entrenamiento ya que este perceptrón solo puede dividir la esfera cuántica en dos partes iguales, por tanto, se hace necesario implementar una capa oculta con el fin de mejorar tanto la precisión como el alcance del perceptrón.

Basados en el código del perceptrón sencillo se realizaron las mejoras pertinentes para que este mismo algoritmo pudiera ser utilizado para implementar el perceptrón multicapa. Se obtuvo que basta con repetir el circuito cuántico y modificar la función de activación de la red para poder realizar la implementación del perceptrón en mención.

Las mejoras y modificaciones que se realizaron en el perceptrón multicapa respecto al código del perceptrón sencillo se muestran la tabla 7.

Tabla 7. Mejoras y modificaciones en el algoritmo para implementar el perceptrón multicapa. (Lozano, 2020)

PERCEPTRÓN SENCILLO	PERCEPTRÓN MULTICAPA
Vectores de entrenamiento: 2	Vectores de entrenamiento: 4
Se corre una única vez el código de normalización y se hace de forma separada para cada vector de entrenamiento.	Se realizó código más general para la normalización de los vectores de entrenamiento y se corre cuatro veces, una por cada vector.
Se crea un circuito cuántico que realiza la tarea de clasificación de dos tipos de datos	Se realiza la ejecución del código del circuito cuántico cuatro veces para clasificar las cuatro áreas referentes a los vectores de entrenamiento.
Se realiza una sola simulación ya que solo hay un circuito cuántico.	Se debe realizar una simulación por cada circuito cuántico generado, es decir, se realizan 4 simulaciones.
Se realiza una interpretación de resultados obteniendo la probabilidad para cada clase.	Se genera la interpretación de resultados para cada circuito cuántico, donde en cada simulación se muestra la probabilidad que tiene el vector de entrada de ser de uno de dos vectores de entrenamiento.
Se aplica función de activación que solo debe reconocer y dato de dos tipos de datos.	Se aplica función de activación exponencial con el fin de determinar la probabilidad que tiene el vector de entrada de ser de uno u otro grupo referente a cada vector de entrenamiento.

Se revisa a cuál de los dos vectores de entrenamiento es más probable que pertenezca el vector de entrada.	Tomando la salida de la función de activación revisa cuál de las probabilidades es la más alta referente a los vectores de entrenamiento. Teniendo en cuenta lo anterior, relaciona con una función condicional la posición del vector de probabilidadescon mayor valor con la clase de acuerdo con la clasificación que se dio inicialmente a los vectores deentrenamiento.

En la figura 34 se muestra un ejemplo del resultado de la ejecución del código del perceptrón multicapa, donde los vectores de entrenamiento se muestran con los colores azul, verde, rojo y amarillo, donde los dos primeros refieren a la clase 0 y los dos últimos a la clase 1, además los puntos referentes a los vectores de entrada se muestran en color celeste si son de clase 0 y morado si fueronclasificados como de clase 1.

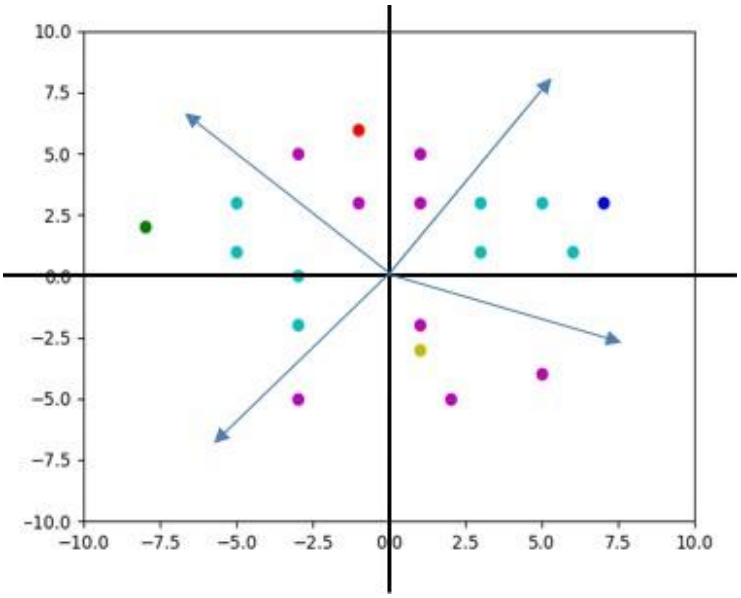

Fig. 34. Ejemplo de la clasificación obtenida por el perceptrón multicapa. (Lozano, 2020)

El código con estas actualizaciones se muestra de forma detallada en el anexo 1.

6. CONCLUSIONES

- El desarrollo de la práctica académica se basó principalmente en el perceptrón cuántico sencillo, pero como se encontró que realiza una clasificación muy limitada y tiende más a errores debido a que solo tiene una neurona y consecuentemente su capacidad de razonamiento es muy limitada se planteó agregar una capa oculta lo que mejoró en gran medida la aplicabilidad de la red neuronal y su exactitud.
- La teoría de superposición fue de vital importancia para el desarrollo del algoritmo en cuestión, inicialmente se tuvieron inconvenientes con la compresión del tema; pero con la asesoría constante del jefe inmediato e investigación se logró comprender e implementar dicha teoría por medio de la esfera cuántica y la posición de los qubits en sus estados de superposición dentro de esta.
- Mediante este primer acercamiento al ambiente laboral se ha logrado un gran aprendizaje y una experiencia muy satisfactoria. Aunque inicialmente el trabajo a realizar fue tomado como un reto debido a los temas que se trabajaron en la practica me siento satisfecho con el trabajo que realicé y con los conocimientos adquiridos. Ahora conozco la importancia del trabajo en equipo de la paciencia y perseverancia necesarias para alcanzar cada uno de los objetivos de la vida profesional y personal.

7. OBSERVACIONES

- Inicialmente se intentó realizar un Simpletrón haciendo uso de lacomputación cuántica, y este proceso duró aproximadamente dos semanas hasta que fueron presentados avances al jefe inmediato. En este punto, Jean Bernier se dio cuenta que me había asignado la actividad por error, yaque hubo una confusión entre el concepto de Simpletrón y perceptrónsimple.
- Como una posible mejora para el algoritmo del perceptrón se plantea implementar la metodología de backpropagation, ya que en el presente documento se realizó de forma no supervisada. Esto ayudaría a que el algoritmo sea más versátil, aunque se dificulta su diseño y digitalización.

8. BIBLIOGRAFÍA

Álvarez, M. F. (2016). *Evolución de la computación cuántica y los retos para la seguridad.* Unisangil.

APD, R. (2019). Tecnicas de la inteligencia artificial. *APD*.

Ballesteros, A. (n.d.). *Neural Networks Framework*. Retrieved from Universidad de Málaga, España: https://redes-neuronales.com.es

Bernier, G. (2018, Octubre 18). *Transformacion digital.* (I. Bernier, Performer) Archivo General de la Nación de Colombia, Bogotá, Cundinamarca, Colombia.

Bernier, G. (n.d.). *Grupo Bernier SAS*. Retrieved from Página Oficial: http://www.berniergroup.com/

Bernier, J. P. (2017, enero). Fermio´s Software. Bogotá, Colombia.

Bibing. (n.d.). Retrieved from El perceptrón: http://bibing.us.es

Bon, R. (2018;2019). Tesi di Laurea. *Quantum algorithm for the implementation of a perceptron*. Padova, Italia.

Burrueco, D. (n.d.). *Interactive Chaos*. Retrieved from Perceptron multicapa: https://www.interactivechaos.com/

Covantec R.L. (2018). *Programación en Python - Nivel básico*. Retrieved from https://entrenamiento-python-basico.readthedocs.io/

Empresarial, G. (2017). Breve hitoria de la nteligencia artificial: el camino hacia la empresa. . *Asesores Depymes*.

Equipo de desarrollo de Qiskit . (2020, 07 17). *Qiskit*. Retrieved from https://qiskit.org/

Everywhere, G. (2018). *Graph Everywhere*. Retrieved from https://www.grapheverywhere.com/

Frutos, A. m. (2017, marzo 18). ¿Qué es la computación cuántica. *Computer hoy*.

innovation, A. (2019, Octubre 22). *Que son las redes neuronales y sus funciones*. Retrieved from ATRIA innovation: ATRIA innovation

Lozano, I. D. (n.d.). *Ingeniero Mecatrónico.* Universidad Autónoma de Bucaramanga, Bucaramanga.

Matich, D. J. (2001, marzo). *Redes neuronales: Conceptos y aplicaciones*. Retrieved from https://www.frro.utn.edu.ar

Microsoft. (2019, 03 19). *Informacion general sobre VIsual Studio*. Retrieved from https://docs.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2019

Microsoft. (2020, 03 05). Retrieved from Tipos de Q#: https://docs.microsoft.com/es-es/quantum/user-guide/language/types

Microsoft. (2020, 02 28). *Conceptos básicos de Q-Sharp*. Retrieved from https://docs.microsoft.com/es-es/quantum/user-guide/basics

Next, V. (2019). Aplicaciones de la inteligencia artificial en la actualidad. *View Next An IBM Subsidiary*.

Ortega, O. (2019, 01 16). *El financiero.* Retrieved from ¿Qué es y para que sirve una computadora cuántica?: https://www.elfinanciero.com.mx/tech/que-es-y-para-que-sirve-una-computadora-cuantica

Palacios, F. J. (n.d.). *Redes neuronales con GNU/Linux*. Retrieved from https://www.ibiblio.org/

Peyrna, R. (2018). principios y metodología de la inteligencia artificial . *Mind meister*.

R, R. M. (2018, 04 26). *ETSI de Sistemas Informáticos – UPM.* Retrieved from Universidad Politecnica de Madrid: https://www.etsisi.upm.es/

Roselló, G. (2017, 03 17). *El gato de Schödinger y la superposición de partículas*. Retrieved from https://gabrielrosselloblog.wordpress.com/

Sáenz, E. (2019, Febrero 13). Youtube. Valencia, España.

Schuld, M., & Petruccione, F. (2014, 12 12). National Institute for Theoretical Physics. *Simulating a perceptron on a quantum computer*. South Africa.

Schuld, M., Fingerhunth, M., & Petruccion, F. (2017, 08). Quantum Reseach Group, School of chemistry and physics. *Implementing a distance-based classifier with a quantum interference circuit*. South Africa.

Torra, V. (2011). La inteligencia artificial. *lychnos Fundacion general CSIC*.

Torres Barrán, A., & Naveiro, R. (2019, 06 04). *Curso de aprendizaje automático para el INE*. Retrieved from Redes Neuronales: http://albertotb.com/curso-ml-R/Rmd/12-nn/12-nn.html#1

9. APÉNDICE

Anexo 1: Código de perceptrón cuántico simple de la primera entrega.

```
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute, BasicAer
from matplotlib import pyplot as plt
from qiskit.visualization import plot_histogram
import math
##*************************************************************************************************************************************##
def classify(test_vector, training_set):
    # Extraer vectores de entrenamiento
    training_vectors = [tuple_[0] for tuple_ in training_set]
    # Inicializar los registros Q y C
    [q, c, ancilla_qubit, index_qubit, data_qubit, class_qubit] = initialize_registers(num_registers=4)
    # Obtener los angulos necesario para representar los datos en el estado cuantico
    angles = get_angles(test_vector=test_vector,training_vectors=training_vectors)
    # Crear el circuito cuantico
    qc0 = create_circuit(q, c, ancilla_qubit, index_qubit, data_qubit, class_qubit, angles, 1, 2)
    qc1 = create_circuit(q, c, ancilla_qubit, index_qubit, data_qubit, class_qubit, angles, 3, 2)
    qc2 = create_circuit(q, c, ancilla_qubit, index_qubit, data_qubit, class_qubit, angles, 3, 4)
    qc3 = create_circuit(q, c, ancilla_qubit, index_qubit, data_qubit, class_qubit, angles, 1, 4)
    # Simular y obtener los resultados
    result0 = simulate(qc0)
    result1 = simulate(qc1)
    result2 = simulate(qc2)
    result3 = simulate(qc3)

    prob0_class0, prob0_class1 = interpret_results(result0.get_counts(qc0),angles,1,2)
    prob1_class0, prob1_class1 = interpret_results(result1.get_counts(qc1),angles,3,2)
    prob2_class0, prob2_class1 = interpret_results(result2.get_counts(qc2),angles,3,4)
    prob3_class0, prob3_class1 = interpret_results(result3.get_counts(qc3),angles,1,4)

    clasefinal = [prob0_class0-0.5,prob1_class0-0.5,prob2_class0-0.5,prob3_class0-0.5,prob0_class1-0.5,prob1_class1-0.5,prob2_class1-0.5,prob3_class1-0.5]
    clase2 = [0,0,0,0,0,0,0,0]
    for var in range (0,len(clasefinal)):
        clase2[var] = 1/(1-math.exp(-clasefinal[var]))
    c = max(clase2);
    c = clase2.index(c)
    prob_class0 = (prob0_class0+prob1_class0+prob2_class0+prob3_class0)/4
    prob_class1 = (prob0_class1+prob1_class1+prob2_class1+prob3_class1)/4

    if c >= 4:
        return 1
    elif c < 4:
        return 0
    else:
        return 'Inconcluso. Resultados: 50/50'
##*************************************************************************************************************************************##
def initialize_registers(num_registers):
    q = QuantumRegister(num_registers)
    c = ClassicalRegister(num_registers)
    # Asignacion de nombre a los qubits
    ancilla_qubit = q[0]
    index_qubit = q[1]
    data_qubit = q[2]
    class_qubit = q[3]
    return (q, c, ancilla_qubit, index_qubit, data_qubit, class_qubit)
##*************************************************************************************************************************************## IMPORTANTE !!!!!!
def get_angles(test_vector, training_vectors):
    angles = []
#__________ VECTOR DE ENTRADA _________________#
    x = test_vector[0]
    y = test_vector[1]
    if(x<0 and y<0):
        alpha = math.pi + abs(math.atan(y/x))
    elif (x<0 and y>0):
        alpha = math.pi - abs(math.atan(y/x))
    elif (x>0 and y<0):
        alpha = 2*math.pi - abs(math.atan(y/x))
    elif (x==0):
        if (y>0):
            alpha = math.pi/2
        else:
            alpha = 3*math.pi/2
    elif (y==0):
        if (x>0):
            alpha = 0
        else:
            alpha = math.pi
```

```
    else:
        alpha =  abs(math.atan(y/x))
    if (x==0 and y==0):
        print("La coordenada no puede ser clasificada")
    else:
        angles.append(alpha)
#__________ VECTOR DE ENTRENAMIENTO 1 __________#
    for d in range (0,4):
        xl = training_vectors[d][0]
        yl = training_vectors[d][1]
        if(xl<0 and yl<0):
            alpha1 = math.pi + abs(math.atan(yl/xl))
        elif (xl<0 and yl>0):
            alpha1 = math.pi - abs(math.atan(yl/xl))
        elif (xl>0 and yl<0):
            alpha1 = 2*math.pi - abs(math.atan(yl/xl))
        elif (xl==0):
            if (yl>0):
                alpha1 = math.pi/2
            else:
                alpha1 = 3*math.pi/2
        elif (yl==0):
            if (xl>0):
                alpha1 = 0
            else:
                alpha1 = math.pi
        else:
            alpha1 =  abs(math.atan(yl/xl))
        if (xl==0 and yl==0):
            print("La coordenada no puede ser clasificada")
        else:
            angles.append(alpha1/2)
    return angles
##*****************************************************************************************##
def create_circuit(q, c, ancilla_qubit, index_qubit, data_qubit, class_qubit, angles, num1, num2):
    # Crear el circuito cuantico vacio
    qc = QuantumCircuit(q,c)
    ##########################################################################################
    #INICIO de la rutina de preparacion del estado
    # Pone los qubits ancilla e indice en superposicion
    qc.h(ancilla_qubit)
    qc.h(index_qubit)
    # Cargando el vector de testeo el cual será clasificado
    qc.cx(ancilla_qubit, data_qubit)
    qc.u3(-angles[0], 0, 0, data_qubit)
    qc.cx(ancilla_qubit, data_qubit)
    qc.u3(angles[0], 0, 0, data_qubit)
    # Las barreras aseguran que el circuito se ejecute de la manera que queremos
    # de otra manera algunas puertas pueden ejecutarse antes de lo deseado
    qc.barrier()
    # Voltea el qubit ancilla. Esto mueve el vector de entrada al estado cero de la ancilla
    qc.x(ancilla_qubit)
    qc.barrier()
    # Cargando el primer vector de entrenamiento
    # Clase 0
    qc.ccx(ancilla_qubit, index_qubit, data_qubit)

    qc.cx(index_qubit, data_qubit)
    qc.u3(angles[num1], 0, 0, data_qubit)
    qc.cx(index_qubit, data_qubit)
    qc.u3(-angles[num1], 0, 0, data_qubit)

    qc.ccx(ancilla_qubit, index_qubit, data_qubit)

    qc.cx(index_qubit, data_qubit)
    qc.u3(-angles[num1], 0, 0, data_qubit)
    qc.cx(index_qubit, data_qubit)
    qc.u3(angles[num1], 0, 0, data_qubit)
    qc.barrier()
    # Voltear el qubit indice > mueve el primer vector de entrenamiento al estado |0> del qubit indice
    qc.x(index_qubit)
    qc.barrier()
    # Cargando el segundo vector de entrenamiento
    # Clase 1
    qc.ccx(ancilla_qubit, index_qubit, data_qubit)

    qc.cx(index_qubit, data_qubit)
    qc.u3(angles[num2], 0, 0, data_qubit)
    qc.cx(index_qubit, data_qubit)
    qc.u3(-angles[num2], 0, 0, data_qubit)
```

```
    qc.ccx(ancilla_qubit, index_qubit, data_qubit)

    qc.cx(index_qubit, data_qubit)
    qc.u3(-angles[num2], 0, 0, data_qubit)
    qc.cx(index_qubit, data_qubit)
    qc.u3(angles[num2], 0, 0, data_qubit)
    qc.barrier()
    # Voltea el qubit de clase para el vector de entrenamiento #2
    qc.cx(index_qubit, class_qubit)
    qc.barrier()
    # FINALIZA la rutina de preparacion de los qubits
    ######################################################
    ##############################################
    # INICIA el clasificador basado en distacia
    # Interviene el vector de entrada con los vectores de entrenamiento
    qc.h(ancilla_qubit)
    qc.barrier()
    # Mide todos los qubits y recupera los datos en los registros clasicos
    qc.measure(q,c)
    # FINALIZA el clasificador basado en distancia
    ##############################################
    return qc
##*************************************************************************************************************************##
def simulate(quantum_circuit):
    # Simulación de ruido
    backend_sim = BasicAer.get_backend('qasm_simulator')
    job_sim = execute(quantum_circuit, backend_sim)
    result = job_sim.result()
    counts = result.get_counts(quantum_circuit)
   # plot_histogram(counts)
   # plt.show()
    return job_sim.result()
##*************************************************************************************************************************##
def interpret_results(result_counts,angles,num1,num2):
    if (abs((abs((2*angles[num1])+(2*angles[num2]))/2)-angles[0]) <= 2.09):
        total_samples = sum(result_counts.values())
        # Define la función lambda que recupera solo resultados donde el qubit ancilla está en el estado |0>
        post_select = lambda counts: [(state, occurences) for state, occurences in counts.items() if state[-1] == '0']
        # Realiza la postselección
        postselection = dict(post_select(result_counts))
        postselected_samples = sum(postselection.values())
        retrieve_class = lambda binary_class: [occurences for state, occurences in postselection.items() if state[0] == str(binary_class)]
        prob_class0 = sum(retrieve_class(0))/postselected_samples
        prob_class1 = sum(retrieve_class(1))/postselected_samples
    else:
        prob_class0 = 1;
        prob_class1 = 1;

    return prob_class0, prob_class1
##*************************************************************************************************************************##
## ************************** EJECUCIÓN DEL PROGRAMA GENERAL ************************** #
n  = 0; n1 = 0; Xo = []; Yo = []; X1 = []; Y1 = [];
print("              ____________ PERCEPTRÓN CUÁNTICO ____________")
print(" ")
while(n == 0):
    if (n1 == 0):
        print("                 ***  ** DATOS DE ENTRENAMIENTO ** ***")
        print("Para iniciar, por favor ingrese las coordenadas del primer punto núcleo de la CLASE 0: ")
        x1 = float(input('x: '))
        y1 = float(input('y: '))
        print("Ahora, por favor ingrese las coordenadas del segundo punto núcleo de la CLASE 0: ")
        x3 = float(input('x: '))
        y3 = float(input('y: '))
        print("Ahora, por favor ingrese las coordenadas del primer punto núcleo de la CLASE 1: ")
        x2 = float(input('x: '))
        y2 = float(input('y: '))
        print("Ahora, por favor ingrese las coordenadas del segundo punto núcleo de la CLASE 1: ")
        x4 = float(input('x: '))
        y4 = float(input('y: '))
    print("Por favor ingrese las coordenadas del punto que desea clasificar: ")
    x = float(input('x: '))
    y = float(input('y: '))
    x_prime = [x, y] # x' = DATO DE ENTRADA A CLASIFICAR

    training_set = [
        ([x1, y1], 0), # PRIMERA COORDENADA DEL DATO DE ENTRENAMIENTO A CLASIFICACIÓN 0
        ([x2, y2], 1), # PRIMERA COORDENADA DEL DATO DE ENTRENAMIENTO A CLASIFICACION 1
        ([x3, y3], 0), # SEGUNDA COORDENADA DEL DATO DE ENTRENAMIENTO A CLASIFICACIÓN 0
        ([x4, y4], 1)] # SEGUNDA COORDENADA DEL DATO DE ENTRENAMIENTO A CLASIFICACIÓN 1
```

```
    print(f"Clasificando x' = {x_prime} con ruido 'Simulator backend'")
    class_result = classify(test_vector=x_prime, training_set=training_set)
    print(f"El vector de entrada x' es de la clase {class_result}\n")
    if (class_result == 0):
        Xo.append(x)
        Yo.append(y)
    else:
        X1.append(x)
        Y1.append(y)
    continuar = input('Para clasificar otro punto digite Y, si desea reentrenar el perceptrón oprima R para finalizar digite E: ')
    if (continuar == 'Y' or continuar == 'y'):
        n1 = 1;
    elif (continuar == 'R' or continuar == 'r'):
        n1 = 0;
    else:
        n = 1;
plt.plot(Xo,Yo,'o',color= 'c')
plt.plot(X1,Y1,'o',color= 'm')

plt.plot(x1,y1,'o',color= 'b')
plt.plot(x2,y2,'o',color= 'r')
plt.plot(x3,y3,'o',color= 'b')
plt.plot(x4,y4,'o',color= 'r')

plt.axis([-1.2,1.2,-1.2,1.2])
plt.show()
```