Certificación de un Generador de Números Aleatorios Cuántico mediante Machine Learning

Julio Marcos Monzón Baldetti

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ciencias y Humanidades

Certificación de un Generador de Números Aleatorios Cuántico mediante Machine Learning

Trabajo de graduación en modalidad de Tesis presentado por
Julio Marcos Monzón Baldetti
para optar al grado académico de Licenciado en Física

Guatemala
2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ciencias y Humanidades

Certificación de un Generador de Números Aleatorios Cuántico mediante Machine Learning

Trabajo de graduación en modalidad de Tesis presentado por
Julio Marcos Monzón Baldetti
para optar al grado académico de Licenciado en Física

Guatemala
2023

Vo. Bo. :

(f) _____
MSc. Mariano Lemus

Tribunal Examinador:

(f) _____
MSc. Mariano Lemus

(f) _____
MSc. Eduardo Alvarez

(f) _____
MSc. Dorval Carías

Fecha de aprobación: Guatemala, 16 de junio de 2023.

Agradecimientos

En primer lugar, quiero agradecer al grupo *Optical Quantum Communications* del Instituto de comunicaciones - Aveiro, Portugal, por brindarme la oportunidad de llevar a cabo esta investigación y por permitirme participar en las reuniones semanales del equipo de trabajo del QRNG. Este trabajo fue realizado gracias al apoyo del proyecto *QuantumPrime* (Referencia: PTDC/EEI-TEL/8017/2020), financiado por la *Fundação para a Ciência e a Tecnologia (FCT)* de Portugal.

Doy las gracias a mi asesor Mariano Lemus, por su constancia y sugerencias para la ejecución de este trabajo. Así mismo, agradezco a Mauricio Ferreira, quien me proveyó la información necesaria para entender el sistema físico así como los datos que utilicé para su análisis.

Finalmente, agradezco a mi familia por su incondicional apoyo durante todo este trayecto.

Índice

Lista de figuras

Lista de cuadros

Lista de acrónimos

ADC Convertidor análogo digital

BS Divisor de haz

CNN Red neuronal convolucional

CTRNG Generador de números aleatorios verdaderos clásico

FNN Red neuronal prealimentada

LCG Generador lineal congruencial

LO Oscilador local

LSTM Memoria larga a corto plazo

PRNG Generador de números pseudoaleatorios

QRNG Generador de números aleatorios cuántico

RCNN Red neuronal recurrente convolucional

RNG Generador de números aleatorios

RNN Red neuronal recurrente

SPD Detector de un solo fotón

SVM Support vector machine

TIA Amplificador de transimpedancia

TRNG Generador de números aleatorios verdaderos

VOA Atenuador óptico variable

Resumen

El presente trabajo constituye un estudio de la aleatoriedad de las secuencias numéricas producidas por un generador de números aleatorios cuántico (QRNG) basado en el principio de fluctuaciones del vacío. Para este fin, se implementaron tres modelos de Machine Learning: una red neuronal prealimentada (FNN), una red neuronal recurrente con arquitectura de memoria larga a corto plazo (LSTM), y una red neuronal recurrente convolucional (RCNN). Estas se sometieron a un proceso de aprendizaje con el propósito de analizar las cadenas numéricas del generador antes y después de una etapa de posproceso, para observar si correlaciones presentes antes de dicho procedimiento se desvanecen luego de aplicarlo. Similarmente, a fin de validar las capacidades predictivas de las redes, se realizó un ataque sobre un generador de números pseudoaleatorios (PRNG) basado en congruencia lineal.

Los resultados del estudio indicaron que las redes lograron detectar una pequeña cantidad de correlaciones en la data antes del posproceso, obteniendo una capacidad predictiva que supera la probabilidad de adivinar el próximo número de la secuencia por poco menos de un 1 %. Se observó que la FNN fue la red más apta en este análisis. Por su parte, las secuencias posprocesadas no mostraron contar con correlaciones, demostrando la resistencia del generador contra las técnicas de aprendizaje aplicadas en este estudio. Por último, el ataque sobre el PRNG verificó que los tres modelos cuentan con las capacidades deseadas, dado que superaron la probabilidad de adivinar de este generador por más de un 80 %.

Abstract

This work constitutes a study of the randomness in the numerical sequences provided by a quantum random number generator (QRNG) based on the principle of vacuum fluctuations. To this end, three Machine Learning models were implemented: a feed-forward neural network (FNN), a recurrent neural network with long short term memory (LSTM) and a recurrent convolutional neural network (RCNN). These models were subjected to a learning process with the purpose of analyzing the numerical strings of the generator before and after a post-processing phase to find out if the correlations observed before said process are still present after it. Similarly, to showcase the predictive qualities of the networks, an attack over a pseudorandom number generator (PRNG) based on linear congruence was performed.

The study's results showed that the networks were able to detect a small amount of correlations in the pre-processed data, achieving a predictive capability that improved over the guessing probability of the next number in the sequence by a little less than 1 %. It was observed that in this particular analysis the FNN had the best performance. On the other hand, the post-processed sequences did not show any correlations, a fact that demonstrates the QRNG's resilience against the learning techniques that were applied. Lastly, it is worth noting that the attack over the PRNG verified that the three models have the desired capabilities, for they were able to beat the guessing probability by more than 80 %.

CAPÍTULO 1

Introducción

Los generadores de números aleatorios, o RNGs por sus siglas en inglés, juegan un papel fundamental en múltiples aplicaciones. Por ejemplo, se utilizan en simulaciones científicas que emplean métodos de Monte Carlo, los cuales requieren para un funcionamiento adecuado RNGs de buena calidad (Harrison, 2010). Los generadores también son esenciales en criptografía, donde sus requerimientos son más estrictos ya que son centrales en la construcción de *llaves criptográficas*, o bien, *claves*, las cuales se usan en protocolos criptográficos cuyo fin es la comunicación secreta en un canal inseguro (Goldreich, 2004). Por lo tanto, la seguridad de dichos protocolos dependerá de que la secuencia del RNG no pueda ser predicha por un adversario.

Por lo general, para la implementación de un RNG se utilizan dos enfoques distintos: los generadores de números pseudoaleatorios (PRNG), y los generadores de números aleatorios verdaderos (TRNG). Las particularidades de cada enfoque se explorarán a detalle más adelante, por el momento, solamente se resalta que un TRNG, por definición, es aquel que produce secuencias de números sin seguir ningún patrón o algoritmo que los vuelva deterministas, o bien, predecibles (Mannalath, Mishra, y Pathak, 2022). Un tipo específico de TRNG es aquel que emplea la naturaleza probabilística intrínseca de los sistemas cuánticos para producir azar, el *Generador de Números Aleatorios Cuántico* o QRNG por sus siglas en inglés (Ferreira, Silva, Pinto, y Muga, 2021). Existen múltiples maneras de llevar a cabo una realización práctica de un QRNG (Mannalath et al., 2022). En el presente estudio, se exploró uno que se basa en el fenómeno cuántico de *fluctuaciones del vacío*. Dicho QRNG ha sido desarrollado por el grupo *Optical Quantum Communications* de la Universidad de Aveiro, situada en Portugal. Este grupo busca proveer servicios informáticos basados en principios cuánticos y entre sus áreas de investigación se encuentra la generación cuántica de números aleatorios.

Actualmente, es necesario certificar un RNG con el fin de asegurar que es apto para las aplicaciones mencionadas, ya que un generador de baja calidad puede comprometer los resultados de simulaciones o la información sensible de usuarios. Debido a esto, existen *suites* de pruebas estadísticas como NIST SP 800-22, Dieharder y TestU01's SmallCrush, que evalúan la aleatoriedad de un RNG. En el pasado, se reportó que el QRNG en cuestión logró pasar todos los tests que involucra cada una de las tres pruebas mencionadas (Ferreira, 2021). Sin embargo, también se ha demostrado que un TRNG que ha logrado superar pruebas estándar de aleatoriedad puede producir secuencias sesgadas que ultimadamente comprometen su calidad (Hurley-Smith y Hernandez-Castro, 2017). Recientemente, se han desarrollado estudios que proponen usar modelos de Machine Learning para analizar RNGs (Kim et al., 2017), (Fan y Wang, 2018), (Truong, Haw, Assad, Lam, y Kavehei, 2018),

(Chai et al., 2019), (Feng y Hao, 2020), (Li et al., 2020). Particularmente, estos estudios se centran en redes neuronales, las cuales proveen un enfoque de evaluación distinto a las pruebas estándar, ya que intentan aprender patrones para predecir el próximo número de una secuencia aleatoria con una probabilidad de éxito mayor a la probabilidad de adivinar.

Con base en esto, en el presente trabajo se desarrolló un análisis de la calidad de la aleatoriedad del QRNG en cuestión empleando técnicas de Machine Learning. Particularmente, con el fin de realizar una comparación de desempeño, se aplicaron tres modelos diferentes: la *red neuronal prealimentada* (FNN), la *red neuronal recurrente* con arquitectura de *memoria larga a corto plazo* (LSTM) y la *red neuronal recurrente convolucional* (RCNN). Cada una tiene principios de aprendizaje distintos, por lo que naturalmente se espera que produzcan resultados variados sobre un mismo problema. Adicionalmente, para validar que las redes neuronales cuentan con la capacidad de detectar correlaciones en secuencias que aparentan ser aleatorias, como parte de la investigación se realizó un ataque sobre un generador pseudoaleatorio de congruencia lineal.

Las partes principales del trabajo se estructuran de la siguiente forma: primero se expone la base teórica tanto de Machine Learning como del funcionamiento del QRNG en el capítulo 4. A continuación, en el capítulo 5 se describe detalladamente el diseño de las redes neuronales así como los aspectos relevantes sobre su entrenamiento y la recolección de resultados. Dichos resultados seguidamente se resumen y ordenan en el capítulo 6 y se analizan como parte del capítulo 7. Finalmente, se presentan las conclusiones del estudio en el capítulo 8.

CAPÍTULO 2

Objetivos

2.1. Objetivo general

Aplicar modelos de Machine Learning a las secuencias producidas por el QRNG basado en fluctuaciones cuánticas del vacío desarrollado por el grupo *Optical Quantum Communications* de la Universidad de Aveiro, a fin de determinar si el generador es resistente a ataques de dichos modelos.

2.2. Objetivos específicos

- Entender la descripción teórica así como práctica del QRNG basado en fluctuaciones del vacío del grupo *Optical Quantum Communications*.
- Implementar las arquitecturas FNN, LSTM y RCNN de redes neuronales para hallar patrones en data secuencial.
- Validar los modelos de Machine Learning demostrando que son capaces de realizar un ataque exitoso contra un PRNG.

CAPÍTULO 3

Justificación

Como sugerido en la introducción, para el buen funcionamiento de una amplia gama de aplicaciones que hacen uso de números aleatorios, se requiere que el generador que los produce sea de buena calidad. Con el propósito de garantizar esto, un RNG puede someterse a pruebas estadísticas que en principio ayudan a detectar la existencia de correlaciones y sesgos en las cadenas aleatorias. Las pruebas particulares mencionadas en la introducción (NIST SP 800-22, Dieharder y TestU01's SmallCrush) no utilizan tests basados en Machine Learning para evaluar los RNGs (Rukhin, Soto, Nechvatal, Smid, y Barker, 2001), (Brown, Eddelbuettel, y Bauer, 2018), (L'Ecuyer y Simard, 2013). Sin embargo, Machine Learning ha demostrado su capacidad de hallar patrones y resolver problemas complejos. Esta disciplina se mantiene en constante crecimiento, y considerando hallazgos como los de Hurley-Smith y Hernandez-Castro (2017), vale la pena tomarla en cuenta en la búsqueda de patrones subyacentes en los números producidos por un RNG. En efecto, múltiples estudios han reafirmado que los tres modelos de redes neuronales propuestos en la introducción pueden utilizarse a fin de detectar interdependencias en secuencias que a primera vista son aleatorias (Kim et al., 2017), (Fan y Wang, 2018), (Truong et al., 2018), (Chai et al., 2019), (Feng y Hao, 2020), (Li et al., 2020). Por lo tanto, es fructífero determinar si el QRNG desarrollado por el grupo *Optical Quantum Communications* es susceptible a las técnicas de Machine Learning presentadas en este estudio puesto que, en el caso de que sea resistente a ellas, le otorgará más confiabilidad a este dispositivo y al uso práctico de QRNGs en general.

CAPÍTULO 4

Marco teórico

4.1. Generación cuántica de números aleatorios

4.1.1. Introducción a generadores de números aleatorios

Un generador de números aleatorios, o RNG por sus siglas en inglés, produce secuencias numéricas que, en principio, carecen de un patrón intrínseco. Los números que se originan de un RNG pertenecen a un conjunto específico, e idealmente, tienen que cumplir con los principios de *uniformidad* e *independencia* (Mannalath et al., 2022). Por uniformidad se entiende que todo número del conjunto debe tener la misma probabilidad de ser producido por el RNG, es decir, son equiprobables. Por ejemplo, si un RNG genera números entre 0 y 9, cada uno tiene una probabilidad de $\frac{1}{10}$ de ser el próximo en la secuencia. Por otro lado, la independencia se refiere a que no deben existir correlaciones entre los números generados. En la actualidad, existen dos tipos de RNGs:

- Generador de números pseudoaleatorios (PRNG): Son aquellos que emplean algoritmos deterministas para generar números a partir de una *semilla*. Dicha semilla puede entenderse como una lista de condiciones iniciales a partir de las cuales el algoritmo realiza sus cálculos. La complejidad de los algoritmos suele ser baja, sin embargo, son capaces de generar secuencias robustas y difíciles de atacar.
- Generador de números aleatorios verdaderos (TRNG): Estos emplean fuentes de entropía física para generar aleatoriedad. Dichas fuentes pueden ser sistemas caóticos, ruido eléctrico y térmico, o bien, sistemas cuánticos (Mannalath et al., 2022). A pesar de tener la ventaja de no basarse en un algoritmo determinista, su implementación es difícil debido a circuitos complejos que son utilizados para extraer la aleatoriedad (Chai et al., 2019).

A pesar de ser deterministas, los PRNGs son una opción versátil para muchas aplicaciones, como simulaciones o predicción del clima. Tienen la ventaja de ser fáciles de implementar, y además sus velocidades de generación son altas. Un ejemplo de PRNG es el *generador lineal congruencial* o LCG. Este hace uso de una relación recursiva para crear una secuencia partiendo de un número semilla. Explícitamente, la fórmula generadora es:

$$X_{n+1} = (aX_n + c) \bmod \mathcal{M} \tag{4.1}$$

donde a es el multiplicador, c el incremento y $\mathcal{M}$ el módulo del generador. Si se cumple que: $\mathcal{M}$ y c son primos relativos, $a-1$ es divisible por todos los factores primos de $\mathcal{M}$ y que $a-1$ sea divisible por 4 cuando $\mathcal{M}$ sea divisible por 4, se generarán números pseudoaleatorios con periodo $\mathcal{M}$ (Li et al., 2020). Por ejemplo, para un LCG que genera números entre 0 y 255, haciendo uso de $a = 25214903917$, $c = 1$ y $\mathcal{M} = 2^{14}$ (que son parámetros empleados por Li et al. (2020)), se obtiene la distribución de probabilidad de la Figura 4.1 para una secuencia de 18000 números pseudoaleatorios. Esta cumple con ser uniforme, en el sentido que en promedio, los números aparecen con una probabilidad de $\frac{1}{256} \approx 0.0039$. Sin embargo, este generador no cumple con el principio de independencia, dado que se tendrán correlaciones entre los números. En efecto, se han realizado ataques exitosos (esto es, se han aprovechado las debilidades del generador para predecir los números pseudoaleatorios con una probabilidad de éxito superior a la probabilidad de adivinar) contra el LCG, los cuales se expondrán más tarde.

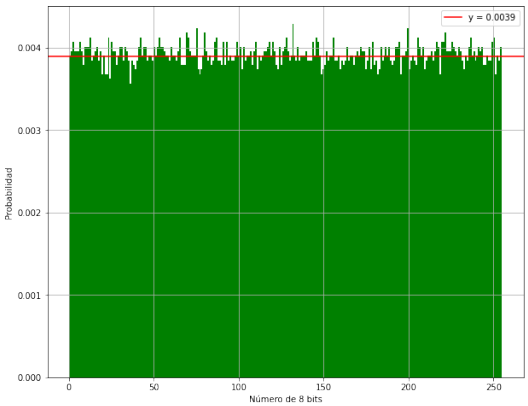

Figura 4.1: Histograma de probabilidad de LCG
Fuente: elaboración propia

La vulnerabilidad de los PRNGs debido a su naturaleza determinista hace que no sean la mejor opción en aplicaciones donde se involucra la privacidad de usuarios y se maneja información sensible. Es por ello que en estos casos es preferible hacer uso de TRNGs debido a su resistencia a ataques. Considerando esto, los TRNGs pueden dividirse en dos subcategorías más, los *clásicos* (CTRNG) y los cuánticos (QRNG). Los QRNGs son aquellos generadores que aprovechan la naturaleza probabilística intrínseca de la mecánica cuántica para producir aleatoriedad, mientras que los CTRNGs abarcan los generadores que utilizan alguna fuente de entropía física clásica i.e. sin relación a sistemas cuánticos. Al basarse solamente en sistemas clásicos, los CTRNGs siguen siendo, en principio, deterministas, lo cual los vuelve vulnerables a ataques. Esta debilidad no se presenta en los QRNGs, mas es de notar que su implementación no es trivial, y lograr velocidades de generación que sean prácticas ha sido un reto en el desarrollo de estos generadores (Mannalath et al., 2022). Adicional a esto, los QRNGs tienen contribuciones clásicas de aleatoriedad que surgen al digitalizar la información que producen, o bien, debido a fluctuaciones de corriente, temperatura o radiación; por lo que es necesario determinar cuánta aleatoriedad es de origen cuántico para solamente extraer este aporte. Cabe mencionar que en casos donde el generador cuántico no produzca una distribución de probabilidad uniforme, es posible someterlo a un *posproceso* en el cual se recupera una distribución uniforme (Ferreira et al., 2021). Naturalmente, aplicar un posproceso agrega dificultad en la implementación del QRNG.

En lo que resta de este capítulo, se hará un enfoque en QRNGs. Específicamente, se expondrá la idea de *entropía* como herramienta para cuantificar la aleatoriedad de un sistema físico y se presentarán algunos tipos de QRNGs. Finalmente, se explorará un QRNG específico, el cual ha sido desarrollado por el grupo *Optical Quantum Communications* de la Universidad de Aveiro, Portugal, y que servirá como objeto de análisis del presente trabajo.

4.1.2. Cuantificación de aleatoriedad

La cuantificación de aleatoriedad se lleva a cabo haciendo uso del concepto de entropía. Esta fue desarrollada inicialmente como una herramienta clave para el entendimiento de sistemas termodinámicos y su evolución. Más tarde, Claude Shannon la introdujo a la teoría de la información al percatarse que a medida que un enunciado tiene una mayor probabilidad de ser verdadero, aporta menos información útil (Blundell y Blundell, 2010). Basándose en esto, definió el *contenido de información* Q de un enunciado como:

$$Q = -k \log_2 P \tag{4.2}$$

donde k es una constante positiva y P es la probabilidad de que el enunciado sea verdadero. Ahora, si se tiene un conjunto de enunciados, cada uno con probabilidad P_i, el contenido de información promedio será:

$$\langle Q \rangle = \sum_i P_i Q_i = -k \sum_i P_i \log_2 P_i \tag{4.3}$$

Es aparente entonces la conexión del contenido de información promedio con la entropía de un sistema termodinámico, puesto que si se utiliza $k = k_B$ y se toma el logaritmo natural en lugar del logaritmo base 2, se recupera la expresión de la entropía de Gibbs:

$$S = -k_B \sum_i P_i \ln P_i \tag{4.4}$$

En el contexto de los RNGs, se utiliza $k = 1$ y la entropía se mide en *bits*. Así, si se tiene una variable aleatoria X con distribución de probabilidad $P_X(x)$ correspondiente, su entropía de Shannon será:

$$H(X) = -\sum_x P_X(x) \log_2 P_X(x) \tag{4.5}$$

Como se mencionó, la expresión (4.5) indica el contenido de información promedio en X. Empleando la técnica de multiplicadores de Lagrange, se determina que la entropía alcanza un máximo cuando $P_i = P = \text{cte}\ \forall\ i$ i.e. en el caso donde los eventos son equiprobables. Recordando el principio de uniformidad, se identifica entonces a la entropía como una medida de la aleatoriedad del sistema, dado que a mayor entropía se tendrá una distribución más uniforme, o bien, aleatoria (Mannalath et al., 2022).

Es posible utilizar una generalización de la entropía de Shannon para medir la aleatoriedad: la entropía de Rényi. Esta se define como:

$$H_\alpha(X) = \frac{1}{1-\alpha} \log_2 \left(\sum_x P_X(x)^\alpha \right) \tag{4.6}$$

donde a α se le conoce como el *orden* de la entropía. Es posible verificar que esta es una medida válida de entropía dado que cuenta con varias propiedades de la expresión de Shannon. Además, utilizando la regla de l'Hôpital es posible verificar que (4.6) tiende a la entropía de Shannon en el límite $\alpha \to 1$. La demostración de varias de las propiedades de esta entropía van más allá del alcance del presente trabajo, para más información puede consultarse Rényi (1961).

Un caso particular se da cuando $\alpha \to \infty$. En este límite, se recupera la *entropía mínima* del sistema (Mannalath et al., 2022):

$$H_\infty(X) = -\log_2 \left[\max_x P_X(x)\right] \tag{4.7}$$

Comparando con la expresión de contenido de información Q, se observa que esta entropía supone una situación donde se extrae la menor cantidad de información posible del sistema, es decir, sirve como una cota inferior para la información que puede extraerse. En términos de aleatoriedad, H_∞ caracteriza al sistema con el valor más bajo de aleatoriedad posible, el "peor de los casos". Es por eso que se emplea para evaluar la aleatoriedad de los RNGs, dado que asumir un valor mayor de entropía da espacio a estimaciones incorrectas que pueden comprometer la calidad del generador. Adicionalmente, la entropía mínima es útil a la hora de *uniformar* una variable aleatoria i.e. convertir una distribución no uniforme en uniforme. Esto se explora a continuación por medio de un ejemplo.

Considere el problema de uniformar la variable aleatoria Y, donde $P(Y = 0) = 0.5$, $P(Y = 1) = 0.25$, $P(Y = 2) = 0.125$ y $P(Y = 3) = 0.125$. Para volverla uniforme, pueden definirse dos eventos nuevos, asociados a una variable aleatoria Z. El evento $Z = 0$ corresponde al caso $Y = 0$, y el evento $Z = 1$ se da cuando $Y \neq 0$. Por ende $P(Z = 0) = P(Z = 1) = 0.5$ y Z tiene una distribución uniforme pues sus eventos son equiprobables. Cada medición de Z puede interpretarse como un bit de información, dado que esta variable toma 0 o 1 como sus valores. Por otro lado, la entropía mínima de Y es:

$$H_\infty(Y) = -\log_2 \left[\max_y P_Y(y)\right] = -\log_2(0.5) = 1 \text{ bit} \tag{4.8}$$

Es decir, como mínimo, puede extraerse un bit de información de Y. Este bit corresponde al que se obtiene en cada medición de Z. Esto da la noción de que si se tiene la variable aleatoria no uniforme X_1, puede construirse una nueva variable aleatoria uniforme X_2 de la cual pueden extraerse $H_\infty(X_1)$ bits. Formalmente, esto se logra empleando un *extractor*, el cual es una función que toma como entradas una variable aleatoria X definida sobre $\{0,1\}^n$ y una semilla U_d uniformemente distribuida en $\{0,1\}^d$. Acá $\{0,1\}^\ell$ representa las cadenas binarias de longitud ℓ. Con estos argumentos, el extractor produce una distribución $\text{Ext}(X, U_d)$ que es ϵ cercana a una distribución uniforme U_m sobre $\{0,1\}^m$, donde ϵ caracteriza el error. Específicamente, a $\text{Ext}(X, U_d)$ se le llama *extractor* (k, ϵ), donde $H_\infty(X) \geq k$ (Reyzin, 2011).

La existencia de un extractor para situaciones más generales que las del ejemplo anterior viene asegurada por el *lema hash sobrante* (Reyzin, 2011). Para establecerlo, es necesario antes definir las *funciones de hash criptográficas* y las *familias universales*. Una función de hash criptográfica, o solamente función de hash, es un mapeo h que toma una cadena de longitud n y le asigna otra de longitud m donde $m < n$, comprimiendo así el mensaje. Se dice que h tiene una *colisión* en x y x' cuando $h(x) = h(x')$. Particularmente, h debe cumplir con ser resistente a colisiones, esto es, dado x, un adversario no debe ser capaz de hallar una colisión x' en un tiempo eficiente. De este modo, las funciones de hash proveen una manera de *autenticar* mensajes. Un aspecto a notar es que las funciones de hash suelen pertenecer a una *familia* $\mathcal{H}$, tal que h_s es el miembro s de la familia (Katz y Lindell, 2020). Se dice que la familia $\mathcal{H}$ de tamaño 2^d es una familia universal de funciones $h : \{0,1\}^n \to \{0,1\}^m$ si para todo $x, y \in \{0,1\}^n$ tal que $x \neq y$, se cumple que (Reyzin, 2011):

$$P_{h \in \mathcal{H}}\left[h(x) = h(y)\right] \leq 2^{-m} \tag{4.9}$$

Con estas definiciones, es posible establecer formalmente el lema hash sobrante.

Lema Hash Sobrante (LHS): Sea X una variable aleatoria tal que $H_\infty(X) \geq k$. Por otro

lado, sea $\mathcal{H}$ una familia universal de funciones de hash de tamaño 2^d que producen cadenas de largo $m = k - 2\log_2(1/\epsilon)$. Entonces

$$\text{Ext}(x, h) = h(x) \tag{4.10}$$

es un extractor $(k, \epsilon/2)$ con longitud de semilla d que provee una distribución $\epsilon/2$ cercana a una distribución uniforme U_m (Reyzin, 2011).

La prueba de esta propiedad puede consultarse en Reyzin (2011), pero lo que se desea resaltar acá es que una familia universal de funciones de hash constituye un extractor que permite uniformar cualquier variable aleatoria.

Ahora bien, puede suponerse un caso donde un atacante posee cierta información E que está correlacionada con la variable aleatoria X, una situación relevante en el estudio de RNGs. Se tendrá entonces una distribución de probabilidad condicionada i.e. dado E, ¿cuál es la probabilidad de X?. En este caso, la expresión (4.7) deja de ser confiable para predecir la entropía mínima de X. A raíz de esto, pueden realizarse 2 definiciones distintas de *entropía mínima condicionada*. Primero se cuenta con la entropía mínima *promedio* de X condicionada a E:

$$H_\infty(X|E) \equiv -\log_2 \mathbb{E}\left[\max_x P\left(X|E=e\right)\right] \tag{4.11}$$

donde $\mathbb{E}$ es un valor esperado, que se toma sobre los posibles valores e de E (Reyzin, 2011). Por otro lado, puede hacerse uso de la entropía mínima condicionada del *peor de los casos* (Haw et al., 2015):

$$H_\infty(X|E) \equiv -\log_2 \left[\max_{e\in\mathbb{R}} \max_{x_i \in X} P_{X|E}(x_i|e)\right] \tag{4.12}$$

Con (4.11) y (4.12) es posible obtener una distribución uniforme asociada a X considerando a un adversario que tiene acceso al ruido clásico del generador con precisión arbitraria (Haw et al., 2015).

4.1.3. Diferentes tipos de QRNGs

A continuación se hace un breve desarrollo de distintos QRNGs que han sido implementados empleando distintos fenómenos físicos que son de naturaleza cuántica.

4.1.3.1. QRNG basado en decaimiento radiactivo

En su forma más simple, este generador emplea una fuente radiactiva y un contador Geiger-Müller (GM). El GM es un detector de radiación ionizante, sensible a radiaciones alfa, beta y gamma. Cuando una partícula ionizante lo atraviesa, genera un pulso (Shultis y Faw, 2016). Al ser expuesto a un flujo constante de radiación, generará pulsos a intervalos desiguales. Esto permite generar números aleatorios a partir de los tiempos de detección. Existen dos métodos para esto:

- Reloj rápido: En este caso, la tasa de detección es más lenta que la frecuencia del reloj. El número aleatorio se toma como la cantidad de ciclos de reloj entre dos detecciones consecutivas.
- Reloj lento: Se trata de una situación donde la tasa de detección es más alta que la frecuencia del reloj. El número aleatorio es la cantidad de detecciones realizadas en un solo ciclo de reloj.

Una de las propiedades de este generador es que produce una distribución de Poisson, por lo tanto, requiere de una etapa de posproceso. Por otro lado, esta clase de QRNG tiene que manipularse con mucho cuidado debido al manejo de substancias radiactivas. Esta radiactividad puede además dañar los detectores, lo cual supone una clara desventaja. Cabe mencionar que el uso del GM establece una limitación en la frecuencia de generación del QRNG, esto debido al *tiempo-muerto* del detector, el cual es un periodo de inactividad del GM luego de cada detección exitosa (Mannalath et al., 2022).

4.1.3.2. QRNG basado en ruido electrónico

La naturaleza cuántica de las partículas que conforman una corriente eléctrica da lugar a ruido electrónico en componentes de circuitos como resistencias y diodos. De dicho ruido es posible extraer aleatoriedad y generar secuencias con métodos de reloj como el QRNG de decaimiento radiactivo, o bien, comparando el ruido con cierto valor umbral de voltaje.

El ruido de un componente electrónico puede dividirse en *ruido de disparo* y *ruido térmico*. El primero surge de la cuantización de la carga, dado que las fluctuaciones de números de ocupación están relacionadas a fluctuaciones en la corriente (Blanter y Büttiker, 2000). Por ende, este es un fenómeno puramente cuántico. Por su parte el ruido térmico se debe a la temperatura ambiental, por lo que su aporte a la entropía del sistema es clásico. Por ende, es necesario determinar cuanta aleatoriedad se debe solamente al ruido de disparo y no al térmico, una tarea complicada que dificulta la implementación de estos QRNGs (Mannalath et al., 2022).

A pesar de esto, ComScire, una empresa que entre sus productos ofrece QRNGs, ha producido generadores que se basan en este principio de ruido de disparo (ComScire, s.f.). Por otro lado, empresas como ID Quantique (IDQuantique, s.f.) y Toshiba (Toshiba, s.f.) se han enfocado en el desarrollo de QRNGs que hacen uso de principios ópticos. En base a esto, a continuación se presentarán algunos QRNGs ópticos.

4.1.3.3. QRNG basado en el camino del fotón

Este dispositivo hace uso de una fuente de un solo fotón, un *divisor de haz*, y 2 detectores de un solo fotón, o SPD por sus siglas en inglés. La fuente emite un fotón, el cual atraviesa un divisor de haz balanceado cuyas salidas de transmisión y reflexión están conectadas cada una a un SPD (usualmente un tubo fotomultiplicador) mediante cables de misma longitud. Denotando el cable del divisor de haz que transporta a un fotón transmitido como *camino 1*, y al que guía a un fotón reflejado como *camino 2*, el estado de un fotón transmitido puede expresarse como $|1\rangle_1|0\rangle_2$. Similarmente, para un fotón reflejado el estado es $|0\rangle_1|1\rangle_2$. Por ende, luego de atravesar el divisor de haz, el estado del fotón puede expresarse como la superposición

$$\frac{|1\rangle_1|0\rangle_2 + |0\rangle_1|1\rangle_2}{\sqrt{2}} \tag{4.13}$$

Al ser detectado por uno de los tubos fotomultiplicadores, este estado colapsa con probabilidad de $\frac{1}{2}$ a $|1\rangle_1|0\rangle_2$ o $|0\rangle_1|1\rangle_2$. De este modo, si se detecta un fotón transmitido (reflejado), esto se interpreta como un 0, y si se trata de un fotón reflejado (transmitido), se trata de un 1.

Este es un generador cuyo principio de funcionamiento es simple y es relativamente fácil de implementar. Tiene además la ventaja de requerir una etapa de posproceso mínima, debido a que el divisor de haz es balanceado. Sin embargo, el tiempo-muerto de los tubos fotomultiplicadores puede llevar a correlaciones en la secuencia aleatoria. Cabe mencionar que estos SPDs no son perfectos, y puede que detecten fotones cuando en realidad no hay. Esto limita la versatilidad de estos QRNGs (Mannalath et al., 2022).

4.1.3.4. QRNG basado en el tiempo de llegada de fotones

Este generador tiene un principio de funcionamiento similar al QRNG basado en decaimiento radiactivo. Utilizando una fuente de un solo fotón, un SPD y un reloj, es posible producir números aleatorios empleando los procedimientos de reloj rápido y reloj lento presentados anteriormente. Considerando que el reloj se reinicia en cada detección, también puede realizarse una implementación donde se le asignan a 2 fotones consecutivos tiempos de llegada t_1 y t_2. Así, si $t_1 > t_2$, el generador produce un 0, y de lo contrario se obtiene un 1.

Dado que los fotones se producen siguiendo una distribución exponencial, los números aleatorios no cumplirán el principio de uniformidad, por ende, estos QRNGs requieren de una fase de posprocesamiento (Mannalath et al., 2022).

4.1.3.5. QRNG basado en fluctuaciones del vacío

Al cuantizar el campo electromagnético, su descripción energética se hace en términos de fotones excitados. Cuando se tienen 0 fotones excitados se tiene el *estado del vacío*, cuyas fluctuaciones dan lugar a una distribución normal con incertidumbre de $\frac{1}{4}$. Al realizar mediciones sobre los llamados *operadores de cuadratura* del campo, será posible obtener números aleatorios.

La obtención de la secuencia aleatoria se realiza mediante el esquema de detección homodina ilustrado en la Figura 4.2. La configuración requiere de un oscilador local (LO), que suele ser un láser; un divisor de haz (BS) balanceado y 2 detectores. El LO ingresa al divisor de haz por una de sus entradas. La otra entrada del divisor está bloqueada con el fin de simular el estado del vacío. Cada una de las salidas del divisor se conecta a un detector, y luego se obtiene la diferencia entre las señales de dichos detectores. Esta última señal permite obtener los números aleatorios (Mannalath et al., 2022).

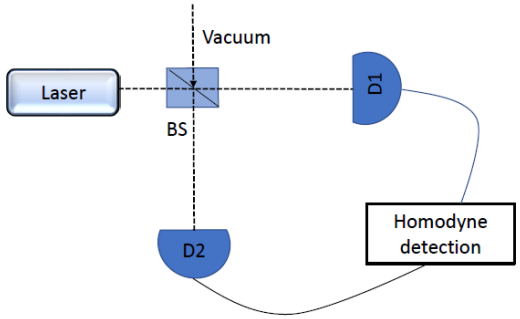

Figura 4.2: Esquema de detección homodina
Fuente: Mannalath et al. (2022)

La descripción anterior es una versión resumida del funcionamiento de este QRNG. A continuación, se hará una descripción más detallada de sus características al explorar una implementación que utiliza fluctuaciones del vacío que ha sido desarrollada en la universidad de Aveiro, Portugal.

4.1.4. QRNG de universidad de Aveiro

4.1.4.1. Cuantización del campo electromagnético y el estado del vacío

Para la cuantización del campo electromagnético se realiza primero el cálculo clásico de la energía. Luego esta se expresa en términos de *variables canónicas*, lo cual permite promoverlas a operadores

y así obtener el *hamiltoniano* del sistema cuántico. La energía clásica es (Gerry y Knight, 2005):

$$H = 2\epsilon_0 V \sum_{\mathbf{k}} \omega_k^2 A_{\mathbf{k}} A_{\mathbf{k}}^* \tag{4.14}$$

donde V es el volumen de la cavidad cúbica donde se modela el campo y $A_{\mathbf{k}}$ es la amplitud del *potencial vectorial*. En este caso solamente se está considerando una posible polarización, de lo contrario habría que incluir una sumatoria adicional que tomaría en cuenta el aporte de la segunda dirección de polarización. El vector de onda $\mathbf{k}$ determina un *modo* del campo electromagnético. Se definen entonces las variables canónicas $q_{\mathbf{k}}$ y $p_{\mathbf{k}}$ al expresar la amplitud del potencial como:

$$A_{\mathbf{k}} = \frac{1}{2\omega_k(\epsilon_0 V)^{1/2}} \left[\omega_k q_{\mathbf{k}} + i p_{\mathbf{k}}\right] \tag{4.15}$$

$$A_{\mathbf{k}}^* = \frac{1}{2\omega_k(\epsilon_0 V)^{1/2}} \left[\omega_k q_{\mathbf{k}} - i p_{\mathbf{k}}\right] \tag{4.16}$$

Al sustituir en la expresión de la energía, la ecuación (4.14) se reduce a:

$$H = \frac{1}{2} \sum_{\mathbf{k}} p_{\mathbf{k}}^2 + \omega_k^2 q_{\mathbf{k}}^2 \tag{4.17}$$

Promoviendo estas cantidades a operadores, el hamiltoniano que describe el sistema cuántico es:

$$\hat{H} = \frac{1}{2} \sum_{\mathbf{k}} \hat{p}_{\mathbf{k}}^2 + \omega_k^2 \hat{q}_{\mathbf{k}}^2 \tag{4.18}$$

O bien, denotando el hamiltoniano correspondiente al vector de onda $\mathbf{k}$ como:

$$\hat{H}_{\mathbf{k}} = \frac{1}{2} \left(\hat{p}_{\mathbf{k}}^2 + \omega_k^2 \hat{q}_{\mathbf{k}}^2\right) \tag{4.19}$$

El hamiltoniano del campo electromagnético se reduce a:

$$\hat{H} = \sum_{\mathbf{k}} \hat{H}_{\mathbf{k}} \tag{4.20}$$

Cada término de la suma de este hamiltoniano corresponde a un oscilador armónico cuántico con masa unitaria. Esto sugiere que para cada modo (i.e. cada $\mathbf{k}$) pueden definirse sus correspondientes operadores aniquilación $\hat{a}_{\mathbf{k}}$ y creación $\hat{a}_{\mathbf{k}}^\dagger$ como:

$$\hat{a}_{\mathbf{k}} = \frac{1}{\sqrt{2\hbar\omega_k}} \left(\omega_k \hat{q}_{\mathbf{k}} + i\hat{p}_{\mathbf{k}}\right) \tag{4.21}$$

$$\hat{a}_{\mathbf{k}}^\dagger = \frac{1}{\sqrt{2\hbar\omega_k}} \left(\omega_k \hat{q}_{\mathbf{k}} - i\hat{p}_{\mathbf{k}}\right) \tag{4.22}$$

Adicionalmente, por cada $\mathbf{k}$ se define el conjunto de eigenestados del operador $\hat{H}_{\mathbf{k}}$ como $\{|n_{\mathbf{k}}\rangle\}_{n=0}^{\infty}$, donde cada uno representa un estado en el cual hay n fotones excitados en el modo $\mathbf{k}$. Dichos estados cumplen con las relaciones usuales del oscilador armónico (Loudon, 2000):

$$\hat{H}_{\mathbf{k}}|n_{\mathbf{k}}\rangle = E_{n_{\mathbf{k}}}|n_{\mathbf{k}}\rangle \tag{4.23}$$

$$E_{n_{\mathbf{k}}} = \hbar\omega_k\left(n_{\mathbf{k}} + \frac{1}{2}\right) \tag{4.24}$$

$$\hat{a}_{\mathbf{k}}|n_{\mathbf{k}}\rangle = \sqrt{n_{\mathbf{k}}}|n_{\mathbf{k}} - 1\rangle \tag{4.25}$$

$$\hat{a}^{\dagger}_{\mathbf{k}}|n_{\mathbf{k}}\rangle = \sqrt{n_{\mathbf{k}} + 1}|n_{\mathbf{k}} + 1\rangle \tag{4.26}$$

Es por estas últimas dos ecuaciones que se interpreta la acción de $\hat{a}_{\mathbf{k}}$ y $\hat{a}^{\dagger}_{\mathbf{k}}$ como la destrucción y creación de un fotón en el modo $\mathbf{k}$ del campo electromagnético, respectivamente. Cabe mencionar que en este esquema se entiende que los fotones se "distribuyen" en la cavidad y no se les ve como partículas. El operador *número de fotones* $\hat{n}$ se define como:

$$\hat{n}_{\mathbf{k}} = \hat{a}^{\dagger}_{\mathbf{k}}\hat{a}_{\mathbf{k}} \tag{4.27}$$

De los conceptos anteriores, es posible construir el estado multimodal del campo electromagnético:

$$|\{n_{\mathbf{k}}\}\rangle \equiv |n_{\mathbf{k}_1}\rangle|n_{\mathbf{k}_2}\rangle|n_{\mathbf{k}_3}\rangle\cdots \tag{4.28}$$

Un estado particular del campo electromagnético es el estado del vacío, en el cual ningún modo tiene fotones excitados (Gerry y Knight, 2005):

$$|\{0\}\rangle = |0\rangle|0\rangle|0\rangle\cdots \tag{4.29}$$

Por otro lado, para cada modo es posible definir los operadores de cuadratura como (Loudon, 2000):

$$\hat{X}_{\mathbf{k}} = \frac{1}{2}\left(\hat{a}^{\dagger}_{\mathbf{k}} + \hat{a}_{\mathbf{k}}\right) \tag{4.30}$$

$$\hat{Y}_{\mathbf{k}} = \frac{1}{2}i\left(\hat{a}^{\dagger}_{\mathbf{k}} - \hat{a}_{\mathbf{k}}\right) \tag{4.31}$$

$\hat{X}_{\mathbf{k}}$ e $\hat{Y}_{\mathbf{k}}$ son formas adimensionales de los operadores canónicos $\hat{q}_{\mathbf{k}}$ y $\hat{p}_{\mathbf{k}}$, respectivamente. Es simple demostrar que los valores esperados de $\hat{X}_{\mathbf{k}}$ e $\hat{Y}_{\mathbf{k}}$, así como sus varianzas, son:

$$\langle\hat{X}_{\mathbf{k}}\rangle = \langle\hat{Y}_{\mathbf{k}}\rangle = 0 \tag{4.32}$$

$$\sigma^2_{\hat{X}_{\mathbf{k}}} = \sigma^2_{\hat{Y}_{\mathbf{k}}} = \frac{1}{4}\left(2n_{\mathbf{k}} + 1\right) \tag{4.33}$$

Particularmente, para el estado del vacío, se tienen las llamadas *fluctuaciones del vacío*, que se predicen en la ecuación (4.33) cuando se hace $n_{\mathbf{k}} = 0$:

$$\sigma^2_{\hat{X}_\mathbf{k}} = \sigma^2_{\hat{Y}_\mathbf{k}} = \frac{1}{4} \tag{4.34}$$

Ahora bien, la función de onda de un oscilador armónico en su estado fundamental (i.e. para el estado del vacío), en la representación de cuadratura es (Ferreira, 2021):

$$\psi_{0_\mathbf{k}}(x) = \left(\frac{2}{\pi}\right)^{1/4} e^{-x^2} \tag{4.35}$$

Por consiguiente, la distribución de probabilidad asociada será:

$$P_{0_\mathbf{k}}(x) = |\psi_{0_\mathbf{k}}(x)|^2 = \sqrt{\frac{2}{\pi}} e^{-2x^2} \tag{4.36}$$

la cual es una distribución gaussiana con media 0 y varianza 1/4. Nótese que esto va de acuerdo a lo indicado por las ecuaciones (4.32) y (4.33).

Por lo tanto, el QRNG busca llevar a cabo mediciones de la cuadratura del estado del vacío, con el fin de obtener la distribución dada por (4.36). Así, luego del posproceso adecuado, será posible obtener secuencias de números aleatorios que cumplan con los principios de uniformidad e independencia.

4.1.4.2. Descripción teórica del QRNG

La presente sección y las siguientes se desarrollan siguiendo la elaboración presentada en Ferreira et al. (2021) y Ferreira (2021). Para más información, pueden consultarse dichas referencias.

El esquema del QRNG se presenta en la Figura 4.3. Este describe un arreglo de detección homodina más detallado que el de la sección 4.1.3.5. El oscilador local (LO) es representado por el operador aniquilación $\hat{a}_{\mathrm{LO}}(t)$, mientras que el estado del vacío se denota $\hat{v}_S(t)$. Una descripción breve de la funcionalidad de los componentes del circuito se presenta a continuación.

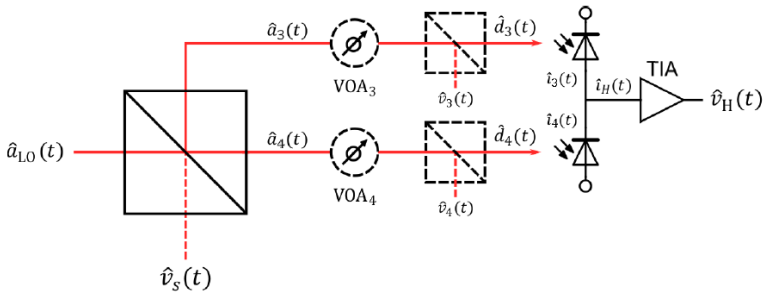

Figura 4.3: Circuito teórico del QRNG
Fuente: Ferreira et al. (2021)

- Divisor de haz (BS): El BS, representado por un cuadrado con una diagonal, refleja y transmite parte de un rayo de luz incidente, de acuerdo a los coeficientes complejos $\mathcal{R}$ y $\mathcal{T}$, respectivamente, donde $|\mathcal{R}|^2 + |\mathcal{T}|^2 = 1$. En el esquema del QRNG, se considera una segunda entrada del divisor la cual se mantiene bloqueada para simular el estado del vacío. Puede verse que, en principio, para el primer BS de la Figura 4.3 con $|\mathcal{R}| = |\mathcal{T}| = \frac{1}{\sqrt{2}}$, se tendrá que:

$$\hat{a}_3(t) = \frac{1}{\sqrt{2}} \left(i\hat{a}_{\mathrm{LO}}(t) + \hat{v}_S(t) \right) \tag{4.37}$$

$$\hat{a}_4(t) = \frac{1}{\sqrt{2}} \left(\hat{a}_{\mathrm{LO}}(t) + i\hat{v}_S(t) \right) \tag{4.38}$$

Donde la unidad imaginaria surge de consideraciones de cambio de fase en una reflexión (Gerry y Knight, 2005). En el contexto del QRNG, las expresiones (4.37) y (4.38) tienen un ligero cambio, pero son fundamentalmente la misma idea.

- Atenuador óptico variable (VOA): Es utilizado para regular las señales ópticas de la detección homodina. Esto es necesario dado que en la práctica los componentes del circuito no son ideales. Los VOA de la Figura 4.3 se modelan como divisores de haz con coeficientes de reflexión y transmisión (Loudon, 2000):

$$\mathcal{R} = i \left(1 - \eta \right)^{1/2} \tag{4.39}$$

$$\mathcal{T} = \eta^{1/2} \tag{4.40}$$

Donde η es la *transmisibilidad* del VOA.

- Fotodiodo: Este dispositivo genera corrientes eléctricas cuando un fotón con energía mayor a cierto umbral incide sobre él (Smith, 2020). En la Figura 4.3, es de notar que los operadores de corriente se definen en términos de los operadores número de los $\hat{d}_i$. Así, por ejemplo para $\hat{i}_3$:

$$\hat{i}_3(t) = q\hat{d}_3^{\dagger}(t)\hat{d}_3(t) \tag{4.41}$$

Donde q es la carga del electrón. En la descripción teórica, a los fotodiodos se les agrega un BS virtual (líneas negras punteadas) para modelar la eficiencia de estos detectores. Los coeficientes del divisor tienen la misma forma que para el VOA, pero a η en este caso se le llama la *eficiencia cuántica* del fotodetector, la cual indica la proporción de fotones incidentes que son detectados (Loudon, 2000).

- Amplificador de transimpedancia (TIA): Convierte una corriente en un voltaje y la amplifica según su ganancia (Horowitz y Hill, 1989).
- Oscilador Local: El LO amplifica la *señal* de la segunda entrada del BS (en este caso, el estado del vacío) para obtener mediciones proporcionales a dicha señal (Loudon, 2000).

El LO y el estado del vacío entran al primer BS de la Figura 4.3, para el cual se define el *desequilibrio* Δ tal que $|\mathcal{R}|^2 = \left(\frac{1}{2} + \Delta\right)$ y por ende, $|\mathcal{T}|^2 = \left(\frac{1}{2} - \Delta\right)$. Las expresiones (4.37) y (4.38) de las señales de salida quedan entonces:

$$\hat{a}_3(t) = i\sqrt{\frac{1}{2} + \Delta}\ \hat{a}_{\mathrm{LO}}(t) + \sqrt{\frac{1}{2} - \Delta}\ \hat{v}_S(t) \tag{4.42}$$

$$\hat{a}_4(t) = \sqrt{\frac{1}{2} - \Delta}\ \hat{a}_{\mathrm{LO}}(t) + i\sqrt{\frac{1}{2} + \Delta}\ \hat{v}_S(t) \tag{4.43}$$

Las señales $\hat{a}_3(t)$ y $\hat{a}_4(t)$ son atenuadas respectivamente por el VOA_3 y el VOA_4, con transmisibilidades η_{VOA3} y η_{VOA4}. Las señales resultantes pasan por los BS virtuales con eficiencias cuánticas η_3 y η_4 tal que $\hat{d}_3(t)$ y $\hat{d}_4(t)$ pueden expresarse como:

$$\hat{d}_3(t) = i\sqrt{\left(\frac{1}{2}+\Delta\right)\eta_3\eta_{\mathrm{VOA3}}}\,\hat{a}_{\mathrm{LO}}(t) + \sqrt{\left(\frac{1}{2}-\Delta\right)\eta_3\eta_{\mathrm{VOA3}}}\,\hat{v}_S(t) + i\sqrt{\eta_3\left(1-\eta_{\mathrm{VOA3}}\right)}\,\hat{v}_{\mathrm{VOA3}}(t) + i\sqrt{1-\eta_3}\,\hat{v}_3(t) \tag{4.44}$$

$$\hat{d}_4(t) = i\sqrt{\left(\frac{1}{2}-\Delta\right)\eta_4\eta_{\mathrm{VOA4}}}\,\hat{a}_{\mathrm{LO}}(t) + \sqrt{\left(\frac{1}{2}+\Delta\right)\eta_4\eta_{\mathrm{VOA4}}}\,\hat{v}_S(t) + i\sqrt{\eta_4\left(1-\eta_{\mathrm{VOA4}}\right)}\,\hat{v}_{\mathrm{VOA4}}(t) + i\sqrt{1-\eta_4}\,\hat{v}_4(t) \tag{4.45}$$

donde $\hat{v}_{\mathrm{VOA3}}(t)$, $\hat{v}_{\mathrm{VOA4}}(t)$, $\hat{v}_3(t)$ y $\hat{v}_4(t)$ son los estados del vacío que corresponden a los divisores de haz asociados a VOA_3, VOA_4, y los fotodiodos, respectivamente. Nótese que acá se hizo uso de los coeficientes de reflexión y transmisión de las expresiones (4.39) y (4.40) para modelar los 4 BSs.

Las señales proceden a ser detectadas por los fotodiodos, los cuales emiten las fotocorrientes representadas por los operadores $\hat{i}_3$ e $\hat{i}_4$. Dichas corrientes son distintas debido a fluctuaciones en el número de detecciones de cada detector (ruido de disparo), las cuales se atribuyen a su vez a las fluctuaciones del vacío. Por lo tanto, se tiene interés en hallar la diferencia $\hat{i}_H$ de las fotocorrientes. Esto se logra aplicando la expresión (4.41):

$$\hat{i}_H(t) = q\left[\hat{d}_3^\dagger(t)\hat{d}_3(t) - \hat{d}_4^\dagger(t)\hat{d}_4(t)\right] \tag{4.46}$$

Idealmente, solamente se realizarían mediciones sobre $\hat{i}_H(t)$, sin embargo, en una implementación práctica existe por lo menos ruido electrónico. La corriente es entonces amplificada y convertida a voltaje mediante el TIA. El observable de voltaje de salida es:

$$\hat{V}_H(t) = G_{\mathrm{TIA}}\left[\hat{i}_e(t) + \hat{i}_H(t)\right] \otimes h(t) \tag{4.47}$$

donde $\hat{i}_e(t)$ representa al ruido electrónico, G_{TIA} es la ganancia del TIA, $h(t) = \delta(t)$ es la función de impulso-respuesta, expresada como delta de Dirac; y $\otimes$ denota la operación de convolución.

La ecuación (4.47) caracteriza al voltaje asociado a las fluctuaciones del vacío. Por lo tanto, mediciones de este observable deben producir, en principio, la distribución Gaussiana predicha en (4.36).

El valor esperado del operador $\hat{V}_H(t)$ puede obtenerse describiendo al LO como:

$$\hat{a}_{\mathrm{LO}}(t) = \hat{\alpha}(t)\exp\left(i\left(\omega_{\mathrm{LO}}t + \theta(t)\right)\right) \tag{4.48}$$

Con $\hat{\alpha}(t) = \sqrt{F + \Delta f_{lo}(t)}$, donde F es el flujo de fotones promedio del LO y $\Delta f_{lo}(t)$ son sus fluctuaciones. Así, es posible llevar a cabo el cálculo del valor esperado del voltaje y este resulta en:

$$\langle \hat{V}_H(t) \rangle = qG_{\mathrm{TIA}}\gamma F \otimes h(t) \tag{4.49}$$

donde:

$$\gamma = \left(\frac{1}{2}+\Delta\right)\eta_3\eta_{\mathrm{VOA3}} - \left(\frac{1}{2}-\Delta\right)\eta_4\eta_{\mathrm{VOA4}} \tag{4.50}$$

Similarmente, es posible calcular la varianza de $\hat{V}_H(t)$. Dicho cálculo se detalla en Ferreira (2021). El resultado es:

$$\sigma_H^2(t=0) = \frac{2\pi}{3} G_{\text{TIA}}^2 \Delta f \left[2\frac{k_B T}{R} + q^2 \beta F + q^2 \text{RIN} \gamma^2 F^2 \right] + \frac{\delta_q^2}{12} \tag{4.51}$$

donde RIN es el ruido de intensidad relativa promedio del ancho de banda del detector, R es la resistencia de carga del TIA, T la temperatura, k_B la constante de Boltzmann, $\frac{\delta_q^2}{12}$ se asocia al error de discretización de un convertidor análogo a digital (no mostrado en la Figura 4.3) y

$$\beta = \left(\frac{1}{2} + \Delta \right) \eta_3 \eta_{\text{VOA3}} + \left(\frac{1}{2} - \Delta \right) \eta_4 \eta_{\text{VOA4}} \tag{4.52}$$

La expresión de la varianza contiene varias contribuciones y no todas son cuánticas. El primer y tercer término entre los corchetes de (4.51) son de origen clásico, mientras que solamente el segundo es de origen cuántico. Es necesario entonces llevar a cabo una estimación de la entropía que está asociada solamente a este último término, para así extraer la cantidad apropiada de aleatoriedad que no comprometa la seguridad del QRNG.

4.1.4.3. Implementación práctica del QRNG

El esquema de la implementación, así como una fotografía del generador, pueden apreciarse en las Figuras 4.4 y 4.5, respectivamente.

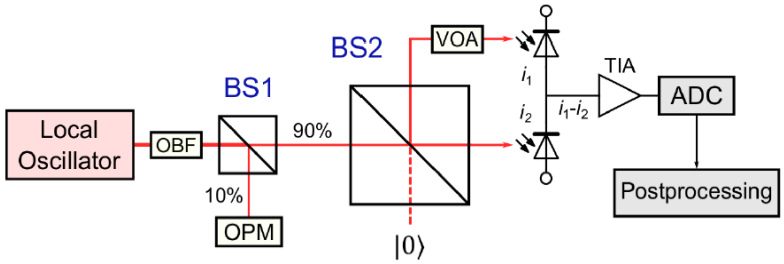

Figura 4.4: Esquema del QRNG implementado
Fuente: Ferreira et al. (2021)

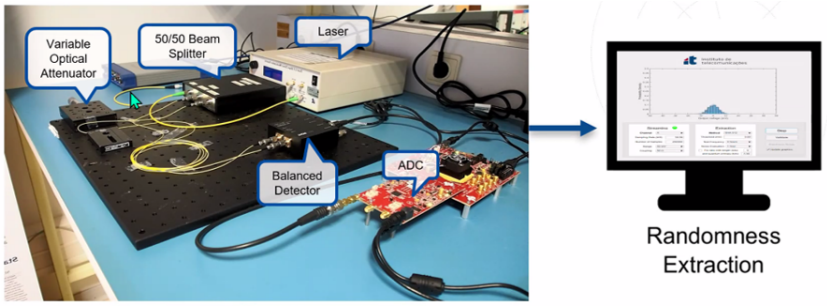

Figura 4.5: Fotografía del QRNG, proveída por el grupo *Optical Quantum Communications*

El oscilador local es un láser de onda-continua de 1550.92 nm, el cual atraviesa un filtro de pasa banda (OBF en la Figura 4.4), que selecciona una frecuencia determinada y atenúa el resto. El LO procede a ser dividido por el divisor de haz BS1, con el fin de utilizar un 10 % de su señal en un

medidor de potencia óptica (OPM en el esquema) para supervisar la potencia del láser. El otro 90 % del LO ingresa en el divisor BS2, el cual recibe en su otra entrada el estado del vacío, denotado por $|0\rangle$. Como se ha mencionado anteriormente, el estado del vacío se logra implementar bloqueando apropiadamente el puerto correspondiente del BS. Una de las dos señales del BS2 es atenuada por un VOA para ajustes del esquema de detección, y ambas señales alcanzan su respectivo fotodiodo para ser detectadas. Dichos fotodiodos se encuentran en un fotodetector balanceado amplificado ThorLabs PDB450C con 45 MHz de ancho de banda de salida. Las fotocorrientes producidas se substraen y son amplificadas por un TIA (que se encuentra también en el ThorLabs PDB450C). El producto de esto es entonces introducido a un convertidor análogo digital Picoscope 6403D de 8 bits con un rango de adquisición de ± 50 mV.

A continuación, sigue el posproceso. Este es necesario por dos principales razones. En primer lugar, la distribución obtenida de las mediciones será gaussiana, por lo que hay que volverla uniforme. Por otro lado, como se indicó en la sección anterior, hay que extraer solamente las contribuciones de aleatoriedad de origen cuántico para que los números del generador sean confiables, sin posibilidad de ser predichos por un atacante con acceso a las contribuciones clásicas.

4.1.4.4. Posproceso

Para el posproceso es necesario determinar la cantidad de entropía asociada a contribuciones cuánticas en el generador, H_Q. De acuerdo con Ferreira (2021) existen dos enfoques para el cálculo de la entropía (siendo el segundo el que actualmente está implementado), los cuales se describen a continuación.

4.1.4.4.1. Entropía de Shannon

En este enfoque primero se construye una distribución uniforme mediante un proceso de agrupamiento de datos (o *binning*) (ver Figura 4.6). Cada bin corresponde a un intervalo de mediciones de voltaje, y se tiene que cumplir que:

$$\int_{-\infty}^{x_1} \rho(x)dx = \int_{x_1}^{x_2} \rho(x)dx = \cdots = \int_{x_l}^{\infty} \rho(x)dx \tag{4.53}$$

Es decir, la integral de la densidad de probabilidad debe tener la misma magnitud en cada intervalo i.e. la probabilidad p_k de que una medición pertenezca al k-ésimo intervalo es constante, $p_k = p \; \forall \; k$.

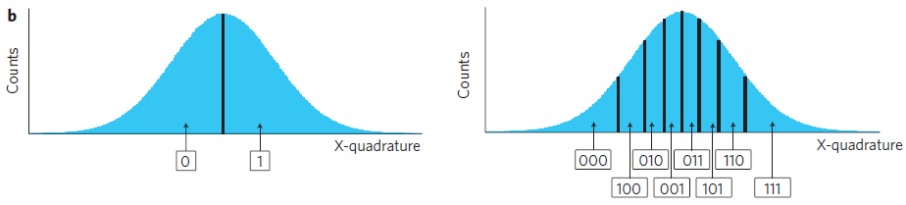

Figura 4.6: *Binning* de la distribución gaussiana
Fuente: Gabriel et al. (2010)

Se tendrán entonces $l + 1$ bins. Se considera el caso particular donde $l + 1 = 2^n$, $n \in \mathbb{N}$. En esta situación, a cada bin se le puede asignar un número de n bits. Por ejemplo, en la Figura 4.6 se tienen las agrupaciones para $n = 1$ y $n = 3$. Si se realiza entonces una nueva medición, y esta pertenece al k-ésimo bin, se genera el número de n bits asociado a dicho bin. Esto da lugar a una distribución uniforme como indica la restricción en (4.53) (Gabriel et al., 2010).

Seguidamente se realiza el cálculo de H_Q. Para ello, primero se determina la entropía total H_T de la distribución uniforme empleando la fórmula de Shannon:

$$H_T = -\sum_{i=1}^{2^n} p_i \log_2 p_i = -\log_2 \frac{1}{2^n} = n \tag{4.54}$$

Similarmente, es necesario hallar la entropía de las contribuciones clásicas H_C. De este modo, H_Q vendrá dada por:

$$H_Q = H_T - H_C \tag{4.55}$$

H_C puede subdividirse en dos contribuciones principales, el ruido electrónico H_E y el ruido excesivo del oscilador local, H_U. Ambas cantidades se determinan por medio de procesos experimentales detallados en Ferreira et al. (2021).

Aunque no se hizo explícito, todas las entropías descritas anteriormente dependen de n. El valor de este parámetro se escoge de tal modo que se optimice la aleatoriedad que puede extraerse de una muestra (acá, la muestra es un dato medido por el ADC que pertenece a la distribución Gaussiana).

Conociendo H_Q, se procede a calcular la *razón de extracción* $\frac{H_Q}{H_T}$ que será útil en la fase de extracción de aleatoriedad.

4.1.4.4.2. Entropía mínima condicionada del peor de los casos

El segundo enfoque consiste en hacer uso de la fórmula (4.12). Esta provee una estimación más adecuada de la entropía que la descrita en la sección anterior dado que toma en cuenta consideraciones asociadas a la discretización realizada por el convertidor análogo digital, lo cual provee la probabilidad máxima de predecir la salida del generador condicionada en que se tiene acceso completo al ruido clásico.

En principio, es necesario modelar un ADC con resolución de n bits. Sea $\delta_d = \frac{R}{2^{n-1}}$ el *ancho de los bins* del convertidor, donde R es su *rango de muestreo*. Si la discretización se centra en el *offset* Δ_d, el *rango de adquisición* será entonces $\left[-R - \Delta_d + \frac{\delta_d}{2}, R - \Delta_d - \frac{3\delta_d}{2}\right]$. Si se definen las variables aleatorias M, que describe una medición; y E, que se refiere al ruido clásico, la distribución de probabilidad condicionada $P_{M|E}(m_i|e)$ es:

$$P_{M|E}(m_i|e) = \begin{cases} \int_{-\infty}^{-R-\Delta_d+\frac{\delta_d}{2}} p_{M|E}(m|e)dm, & i = -2^{n-1} \\ \int_{m_i-\Delta_d-\frac{\delta_d}{2}}^{m_i-\Delta_d+\frac{\delta_d}{2}} p_{M|E}(m|e)dm, & -2^{n-1} < i < 2^{n-1}-1 \\ \int_{R-\Delta_d-\frac{3\delta_d}{2}}^{\infty} p_{M|E}(m|e)dm, & i = 2^{n-1}-1 \end{cases} \tag{4.56}$$

donde $p_{M|E}$ es la función de densidad de probabilidad. Los casos extremos de (4.56) asociados a los bins $i = -2^{n-1}$ e $i = 2^{n-1} - 1$ consideran mediciones que se encuentran fuera del rango de adquisición del ADC, las cuales se mapean a los bins extremos del discretizador.

Ahora bien, tanto M como E tienen una distribución normal. Por ende, puede demostrarse que (Ferreira, 2021):

$$p_{M|E}(m|e) = \frac{1}{\sqrt{2\pi\sigma_Q^2}} e^{-\frac{1}{2}\left(\frac{m-e}{\sigma_Q}\right)^2} \tag{4.57}$$

donde $\sigma_Q^2 = \sigma_M^2 - \sigma_E^2$. Con la densidad de probabilidad es posible llevar a cabo las integraciones en (4.56). Contando con los valores de probabilidad, puede aplicarse entonces la fórmula (4.12), la cual provee la siguiente expresión para la entropía (Ferreira, 2021):

$$H_\infty(M|E) = -\log_2\left[\text{máx}\left\{\frac{1}{2}\left[1+\text{erf}\left(\frac{e_{\text{máx}} - R + \Delta_d + \frac{3\delta_d}{2}}{\sigma_Q\sqrt{2}}\right)\right], \text{erf}\left(\frac{\delta_d}{2\sigma_Q\sqrt{2}}\right)\right\}\right] \tag{4.58}$$

donde $\text{erf}(x)$ es la función error y $e_{\text{máx}} = 5\sigma_E$ es la *excursión máxima del ruido clásico*. Con esta entropía se calcula la razón de extracción y se avanza a la siguiente etapa del posproceso.

4.1.4.4.3. Extracción de aleatoriedad

Finalmente se aplica el extractor de aleatoriedad (ver sección 4.1.2) del cual, en principio, se obtiene una distribución uniforme sin correlaciones o sesgos a partir del muestreo del ADC, el cual sí puede contar con estas imperfecciones. En Ferreira et al. (2021) se indica el uso de dos extractores, siendo el primero la *función de hash criptográfica SHA-512*. Esta recibe una secuencia de longitud n y produce otra con largo $m = 512$ (Chaves, Kuzmanov, Sousa, y Vassiliadis, 2006). Para determinar el valor de n apropiado para el QRNG se emplea la relación

$$n = \left\lceil 512\frac{H_T}{H_Q}\right\rceil \tag{4.59}$$

El otro extractor indicado es el *extractor Toeplitz-hashing* (Ferreira et al., 2021). Este hace uso de *matrices de Toeplitz* para hacer el hash de la secuencia por medio de multiplicación matricial (Krawczyk, 1995). En el contexto del QRNG, la secuencia de entrada de nuevo tiene longitud n, la cual es establecida por el diseñador de tal modo que no sea tan pequeña para evitar llevar a cabo muchas veces el algoritmo de extracción, o muy grande, dado que daría lugar a matrices de extensas dimensiones que aumentan el costo computacional. En este caso, la longitud de la secuencia de salida viene dada por

$$m = \left\lfloor np\frac{H_Q}{H_T}\right\rfloor \tag{4.60}$$

donde p es un valor entre 0 y 1 que se utiliza para emplear solamente una fracción de la razón de extracción $\frac{H_Q}{H_T}$ debido a que H_Q pudo ser sobrestimada. El uso de extractores distintos lleva a diferentes rapideces de generación del QRNG, las cuales ultimadamente afectan lo práctico que es la implementación.

Después de aplicar alguno de los dos extractores mencionados, se cuenta con secuencias que, en principio, son completamente aleatorias. Por lo tanto, están listas para ser utilizadas en las aplicaciones que sean viables dependiendo de la rapidez de generación del QRNG.

4.2. Machine Learning en criptografía

4.2.1. Machine Learning

Machine Learning es una disciplina donde estadística, inteligencia artificial y ciencia computacional convergen, con el fin de otorgar a una computadora la capacidad de realizar predicciones por medio de un modelo matemático de aprendizaje, usualmente llamado *modelo de machine learning*. Esto se logra mediante un proceso de entrenamiento en el cual el modelo de machine learning evoluciona para mejorar su capacidad predictiva. Este entrenamiento es posible llevarlo a cabo de diversas maneras dependiendo de la aplicación en la que se trabaje. Particularmente, existe el *Aprendizaje supervisado*, donde el entrenamiento consiste en proveerle al ordenador un conjunto de entradas con sus correspondientes salidas, o bien, *inputs* y *outputs*. A partir de un algoritmo de machine learning, el modelo logra entonces aprender patrones para que dado un *input* no visto antes sea capaz de predecir el *output* correspondiente con un grado adecuado de precisión.

El aprendizaje supervisado se emplea generalmente en problemas de clasificación (Müller y Guido, 2016), donde se tiene un listado de categorías a las cuales puede pertenecer un *input*. El objetivo es entonces determinar a qué categoría corresponde. También existen problemas de regresión, donde se busca determinar los parámetros adecuados que maximicen la exactitud de las predicciones de un modelo matemático.

Se desea que un modelo de machine learning sea capaz de *generalizarse*, esto es, que pueda realizar predicciones precisas sobre *inputs* nuevos que no pertenecen a la *data* con la que fue entrenado. La calidad de dicha generalización dependerá directamente de las características de la fase de entrenamiento. En principio, en dicha fase es recomendable contar con un conjunto de data que sirva como una muestra representativa de la población que se desea estudiar. Cada instancia dentro de la data debe dividirse en un par de entrada y salida. Posteriormente, la data en su totalidad se divide en tres subconjuntos:

- Conjunto de entrenamiento: Es aquel que se utiliza directamente para el entrenamiento del modelo de machine learning. A partir de este se identifican patrones útiles así como parámetros matemáticos para llevar a cabo predicciones.
- Conjunto de validación: Este se emplea para determinar valores apropiados para los *hiperparámetros* del modelo de aprendizaje. Para esto, se evalúa sobre este conjunto la precisión del modelo para establecer manualmente los hiperparámetros durante el entrenamiento.
- Conjunto de prueba: Este se utiliza para verificar la precisión del modelo durante y al final del entrenamiento, para observar si está logrando mejorar su capacidad predictiva. Esto se logra comparando el *output* del modelo con el real.

Por otro lado, para que un modelo logre generalizarse hay que evitar sobre-entrenarlo, lo cual lleva a un *overfit*. El *overfit* consiste en que el modelo se ajusta a las pequeñas peculiaridades del conjunto de entrenamiento o de la muestra. Esto implica que tendrá problemas a la hora de realizar predicciones sobre la población dado que ésta puede no contar con los detalles más precisos de la muestra. Esto puede entenderse como si el modelo solamente memoriza el conjunto de entrenamiento y no aprende patrones relevantes de él (Nielsen, 2015). El problema opuesto es un *underfit*. En este caso el modelo no logra acoplarse a variaciones en la data, por lo que falla al intentar generalizarse.

El concepto de *overfit* es lo que promueve la creación del conjunto de validación, dado que puede darse un *overfit* del modelo respecto de la data del conjunto de prueba si se desea emplear éste para determinar los hiperparámetros.

El modelo de machine learning más apropiado para llevar a cabo una tarea variará entre aplicaciones. Algunos de los modelos más conocidos son los *k vecinos más cercanos*, los *Árboles de decisión*

y las *Redes neuronales*, entre otros. Cada uno de ellos tiene su forma particular de ser entrenado (Müller y Guido, 2016).

4.2.2. Machine Learning en criptoanálisis

Criptoanálisis es una rama de la criptología que consiste en el análisis de protocolos criptográficos con el fin de evaluar su seguridad. Las técnicas de criptoanálisis se utilizan para intentar desencriptar información que ha sido encriptada por medio de un algoritmo basado en el protocolo en cuestión. Esto se logra generalmente analizando los *outputs* que da el algoritmo para detectar patrones que permitan hallar la clave con la que se llevó a cabo el cifrado o bien, desencriptar el mensaje sin haber obtenido nunca la clave (Meraouche, Dutta, Tan, y Sakurai, 2021).

La noción de detectar patrones en los *outputs* de los algoritmos de cifrado sugiere que es posible aplicar técnicas de machine learning para llevar a cabo estudios de criptoanálisis. En efecto, se ha logrado hacer uso de redes neuronales básicas para detectar código encriptado en programas. Por otro lado, un tipo particular de redes neuronales, conocidas como *redes neuronales convolucionales*, o CNN por sus siglas en inglés, se han empleado para obtener la llave secreta de un circuito criptográfico. Cabe mencionar que otros estilos de redes neuronales, como la *red neuronal recurrente* (RNN), o la red de Hopfield han sido utilizadas con éxito en estudios de criptoanálisis. Adicional a esto, se ha hecho uso del modelo de *SVM* (Support Vector Machine) con resultados variados (Zolfaghari y Koshiba, 2022).

Tanto las CNN como las RNN mencionandas anteriormente constituyen modelos de Deep Learning que es un tipo de Machine Learning. Se ha observado que los modelos de Deep Learning son más versátiles que los que son solamente modelos de Machine Learning en lo que es criptoanálisis (Zolfaghari y Koshiba, 2022). A continuación se explorarán las redes neuronales, y también se hará una pequeña nota sobre los SVM.

4.2.2.1. Redes neuronales

La elaboración posterior se basa en su mayoría en Nielsen (2015). Para información más detallada puede consultarse dicha referencia.

Las redes neuronales surgen del concepto de *neurona*. La neurona es el componente básico de una red: recibe *inputs* y genera un *output* que alimenta a otras neuronas.

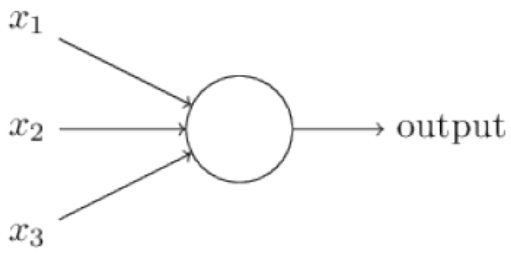

Figura 4.7: Representación gráfica de neurona
Fuente: Nielsen (2015)

En la Figura 4.7 los *inputs* son x_1, x_2 y x_3. Estos, en principio, pueden tomar solamente valores entre 0 y 1. Sin embargo, existen modelos donde este no es el caso. Por simplicidad notacional, estos *inputs* se toman como las componentes de un vector $\mathbf{x}$:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{pmatrix} \tag{4.61}$$

El cálculo del *output* de la neurona se lleva a cabo realizando una suma ponderada de los *inputs* y aplicando una función. Para esto se utilizan pesos $w_1, w_2, ...$ que corresponden a cada *input*, y un sesgo b. La suma ponderada es entonces:

$$\sum_j w_j x_j + b \tag{4.62}$$

Si se define un vector de pesos $\mathbf{w}$ de manera similar a como se definió $\mathbf{x}$, la expresión anterior queda:

$$\mathbf{w} \cdot \mathbf{x} + b \tag{4.63}$$

donde $\cdot$ representa al producto punto entre 2 vectores. A continuación se aplica una función f al resultado de la suma ponderada. A f se le conoce como *función de activación*. El *output* o *activación* de la neurona resulta ser:

$$a = f\left(\mathbf{w} \cdot \mathbf{x} + b\right) \tag{4.64}$$

Como se deduce de la elaboración anterior, los parámetros que determinan el *output* de la neurona son las componentes de $\mathbf{w}$ y b. Sus valores se inicializan aleatoriamente, y mediante el proceso de entrenamiento adquieren los valores óptimos que perfeccionan el *output* global de la red neuronal. Nótese que una red neuronal "simple" puede llegar a tener miles de neuronas. Cada una de estas neuronas cuenta con sus propios pesos y su sesgo. Por ende, en el proceso de entrenamiento de una red usual, es necesario entrenar hasta decenas de miles de parámetros para el funcionamiento óptimo de la red.

Un aspecto relevante del cálculo de a es la función de activación. Su forma puede omitirse durante el desarrollo de la teoría dado que las fórmulas pertinentes son independientes de su expresión explícita. En la práctica, la calidad de una red neuronal puede variar con la función de activación utilizada. A continuación se listan algunas de las funciones de activación empleadas con mayor frecuencia en la elaboración de redes neuronales:

- Función sigmoide:
$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4.65}$$
- Tangente hiperbólica:
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{4.66}$$
- Rectificador o ReLU:
$$\text{ReLU}(z) = \text{máx}\{0, z\} \tag{4.67}$$
- Softmax: Considere el vector $\mathbf{z} = (z_1, z_2, ..., z_r)$. La función softmax es:
$$\text{Softmax}(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{4.68}$$

Una gráfica de la función sigmoide y ReLU se presentan en las Figuras 4.8 y 4.9, respectivamente. La gráfica de $\tanh(z)$ se obtiene de un corrimiento de la función sigmoide y aplicando transformaciones de estiramiento/compresión. Es de notar que tanto la tangente hiperbólica como ReLU no se limitan al intervalo $[0, 1]$, es por esto que se comentó anteriormente que existen instancias donde los valores de las x_i no pertenecen a dicho intervalo.

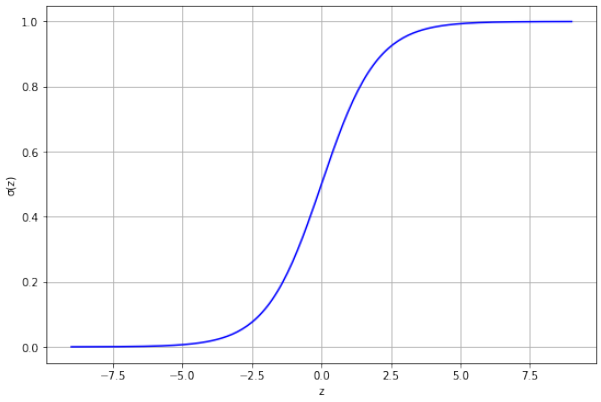

Figura 4.8: Función sigmoide

Fuente: elaboración propia

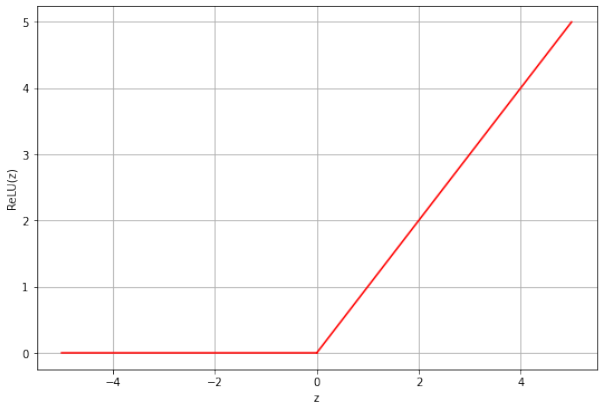

Figura 4.9: Función ReLU

Fuente: elaboración propia

Comprendiendo el funcionamiento de una sola neurona, es posible definir una red neuronal. En términos sencillos, se trata de un conjunto de neuronas organizadas en *capas* que se comunican entre sí. Por comunicación se entiende que el *output* de una neurona es utilizado como *input* de otras. La manera en que las capas están conectadas varía entre distintas arquitecturas de red. Sin embargo, la ilustración que suele ser más general para representar una red neuronal es la que se observa en la Figura 4.10. En dicha figura se identifican tres tipos de capas:

- Capa de *inputs*: Es la primera capa de la red. Acá se introduce el *input* de una instancia de la data.
- Capa de *outputs*: Es la última capa. En ella se obtiene el *output* global de la red. Puede tener múltiples neuronas.
- Capas ocultas: Son las capas intermedias entre las capas de *inputs* y *outputs*. Su número puede variar.

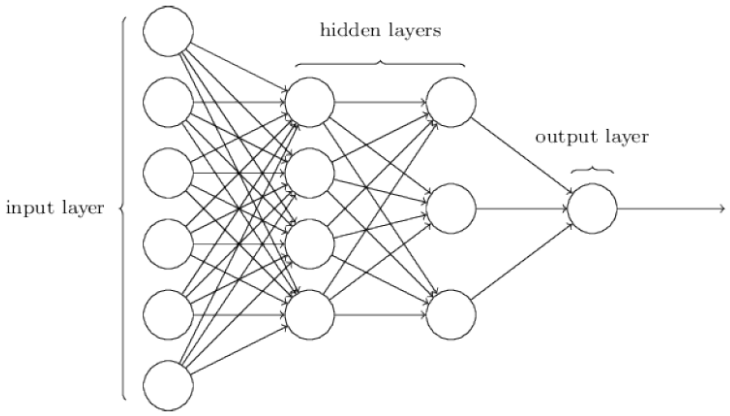

Figura 4.10: Red neuronal prealimentada
Fuente: Nielsen (2015)

Otro aspecto a notar de la Figura 4.10 es que en la red no existen ciclos, esto es, gráficamente toda conexión va de izquierda a derecha y ninguna regresa a una capa anterior de la red. A esto se le conoce como una *red neuronal prealimentada* o FNN. Cuando se permiten ciclos, se tiene una red neuronal recurrente.

Conociendo los conceptos principales, se prosigue a estudiar el proceso de entrenamiento de una red neuronal para obtener los pesos y sesgos que optimicen la red. Para esto, se define el *vector de outputs deseados* $\mathbf{y}(x)$ de un *input* x. Acá x representa un *input* de entrenamiento y no los *inputs* de una neurona particular. Con esto en mente, las componentes de $\mathbf{y}(x)$ son las activaciones que se desea que las neuronas de la última capa de la red obtengan para el *input* x. Por otro lado, se tiene el *vector de activaciones* $\mathbf{a}$ de la última capa. Este contiene la información del *output* (correspondiente a x) que la red logra con sus pesos y sesgos actuales.

Se define entonces una *función de costo* $C(w, b)$ la cual cuantifica qué tan bien realiza la red predicciones sobre el conjunto de entrenamiento. Nótese que el argumento de C involucra a todos los pesos y sesgos de la red. Una característica de la función de costo es que tiene que ser no negativa. Adicional a esto, se desea que $C \to 0$ a medida que las predicciones de la red mejoren. Por lo contrario, C debe crecer si las predicciones de la red son insatisfactorias. Existen varias funciones que cumplen con estas características, siendo la función cuadrática de costo la más elemental:

$$C(w, b) = \frac{1}{2n} \sum_x \|\mathbf{y}(x) - \mathbf{a}\|^2 \tag{4.69}$$

Acá n es el número de ejemplos de entrenamiento y $\|\cdot\|$ denota la norma de un vector. Es fácil verificar que esta función es no negativa, y que cuando $\mathbf{a} \to \mathbf{y}(x)\ \forall\ x \Rightarrow C \to 0$.

Es evidente entonces que para que la red neuronal tenga un buen desempeño es necesario minimizar la función de costo. Para ello, se emplea cálculo multivariable, específicamente, el vector gradiente de C, ∇C, definido como:

$$\nabla C = \left(\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b} \right) \tag{4.70}$$

Nótese que las componentes de ∇C incluyen las derivadas parciales de C respecto de todo peso y sesgo, pero por simplicidad notacional la expresión anterior no hace esto explícito.

Ahora bien, un pequeño cambio ΔC de la función de costo puede expresarse (aproximadamente) de la siguiente manera:

$$\Delta C \approx \frac{\partial C}{\partial w}\Delta w + \frac{\partial C}{\partial b}\Delta b = \nabla C \cdot \Delta u \tag{4.71}$$

donde $\Delta u = (\Delta w, \Delta b)$ es el vector que indica la dirección en la que se está haciendo el cambio en la función de costo. Recordando que $\Delta u = \nabla C$ define la dirección en la que la función incrementa con mayor rapidez, si se desea minimizar la función de costo de manera óptima, se hacen cambios ΔC en la dirección de $-\nabla C$:

$$\Delta C = -\nabla C \cdot \eta \nabla C = -\eta \|\nabla C\|^2 \tag{4.72}$$

donde η es la *tasa de aprendizaje*, un hiper-parámetro de la red que es una medida de los cambios en Δu. Esta tasa tiene que ser pequeña para que la expresión de ΔC en términos de Δu sea válida; pero no tan pequeña para evitar que el entrenamiento lleve mucho tiempo. Además, para que $\Delta C < 0$ debe cumplirse que $\eta > 0$. Con base en esto, se observa que para minimizar la función de costo se tienen que actualizar los pesos y sesgos de la siguiente manera:

$$w \to w' = w - \eta \frac{\partial C}{\partial w} \tag{4.73}$$

$$b \to b' = b - \eta \frac{\partial C}{\partial b} \tag{4.74}$$

Aplicando estas actualizaciones de los pesos y sesgos repetidamente hará que, en principio, eventualmente se llegue a un mínimo de la función de costo. A esta idea de emplear el gradiente de $C(w, b)$ para minimizarla se le conoce como *descenso de gradiente* y es un algoritmo ampliamente utilizado para el entrenamiento de redes neuronales. Por lo general, se utiliza una variación conocida como *descenso de gradiente estocástico* que se basa en el mismo principio pero que acelera notablemente la fase de entrenamiento. Básicamente, se divide el conjunto de entrenamiento en *mini-lotes*, cada uno con m ejemplos de entrenamiento, denotados por $X_1, X_2, ..., X_m$. Con estos es posible estimar el valor del gradiente aplicando un promedio:

$$\frac{1}{m}\sum_j \nabla C_{X_j} \approx \nabla C \tag{4.75}$$

Por lo tanto, la regla de actualización de pesos y sesgos queda:

$$w \to w' = w - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w} \tag{4.76}$$

$$b \to b' = b - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b} \tag{4.77}$$

Se entrena entonces con la data de cada mini-lote, actualizando en cada uno el valor de los pesos y sesgos. Se completa un *epoch* de entrenamiento al haber recorrido todos los mini-lotes. Cabe mencionar que el número de *epochs* empleados para entrenar constituye otro hiperparámetro de la red.

Esta elaboración muestra entonces cómo se logra entrenar a la red neuronal. No obstante, se omitió un paso esencial y este es el cálculo de las componentes de ∇C i.e las derivadas parciales de $C(w, b)$ respecto de cada peso y sesgo en la red. El algoritmo que permite realizar este cálculo de manera eficiente se conoce como *Algoritmo de propagación hacia atrás* o *Retropropagación*. Sus detalles se omiten en el presente trabajo, solamente se menciona con el fin de aclarar este agujero en el desarrollo del descenso de gradiente. Para más información, véase capítulo 2 de Nielsen (2015).

En principio, el algoritmo de descenso de gradiente puede aplicarse a una red neuronal para obtener resultados satisfactorios. No obstante, es posible mejorar el proceso de aprendizaje de la red mediante una amplia diversidad de técnicas. En primer lugar, puede que una función de costo específica logre que el aprendizaje de la red mejore. Esto porque algunas funciones de activación pueden causar que las derivadas parciales de $C(w, b)$ tiendan a 0 para ciertos valores (como el caso de la función sigmoide o la tangente hiperbólica). Una alternativa a la función cuadrática de costo que no cuenta con estos problemas es la *función de entropía cruzada*:

$$\sum_j y_j \ln a_j + (1 - y_j) \ln(1 - a_j) \tag{4.78}$$

Esta evita problemas de desaceleración en el proceso de aprendizaje que surgen en en el caso de la función cuadrática.

Por otro lado, muchas redes neuronales sufren de *overfit* debido al alto número de parámetros que las constituyen. Este problema se detecta al ver cómo evoluciona la precisión de la red neuronal sobre los conjuntos de prueba y validación. Existe *overfit* si a partir de cierto *epoch* del entrenamiento la precisión deja de mejorar en estos conjuntos, pero según el conjunto de entrenamiento, la función de costo sigue acercándose a cero. Esto implica que la red solamente está aprendiendo las pequeñas peculiaridades del conjunto de entrenamiento y no logra generalizarse a data no vista antes.

Para evitar el *overfit* pueden utilizarse técnicas de *regularización*. Estas consisten en evitar que los pesos puedan tener valores muy grandes, esto porque al contar con pesos pequeños, una pequeña variación en un *input* no tendrá un efecto sobresaliente en el *output*. Esto es relevante dado que implica que si un *input* particular del conjunto de entrenamiento cuenta con mucho ruido, esto es, peculiaridades respecto del resto de *inputs* de entrenamiento, la red no será capaz de acoplarse a estas peculiaridades. Más bien, la red solamente podrá ajustarse a características que comparten la mayor parte de los elementos del conjunto. Por lo tanto, se reduce el efecto del *overfit*. Para regularizar la red, se modifica la función de costo agregándole un término adicional. Si $C_0(w, b)$ representa la función de costo original, entonces la función de costo regularizada será:

$$C(w, b) = C_0(w, b) + \frac{\lambda}{2n} \sum_w w^2 \tag{4.79}$$

donde λ es el *parámetro de regularización* tal que $\lambda > 0$ y es otro hiperparámetro de la red. Esta modificación de la función de costo se conoce como *Regularización L2*. Al agregar la suma de los cuadrados de los pesos, se tendrá la tendencia a disminuir el valor de los w para minimizar el costo. No obstante, se evita que los pesos tiendan a cero mediante el parámetro de regularización, este indica qué término de la función de costo regularizada tiene mayor importancia, ya sea $C_0(w, b)$ o la suma del cuadrado de los pesos.

Por otro lado, para reducir el *overfit* puede emplearse el *Dropout*. Esta técnica consiste en borrar temporalmente neuronas de las capas ocultas de la red de manera aleatoria mientras se entrena. El efecto de esto es similar a entrenar redes distintas y al final promediar sus predicciones.

Cabe mencionar que el *overfit* puede evitarse también utilizando un conjunto de data de mayor tamaño. Sin embargo, esto no suele ser posible porque en muchas aplicaciones la obtención de la

data puede ser muy complicada (Nielsen, 2015).

Terminada la exploración de las técnicas que mejoran el proceso de aprendizaje de la red neuronal, se concluye el vistazo general que se ha dado a este modelo de machine learning. A continuación se desarrollan brevemente arquitecturas de redes neuronales específicas de relevancia actual.

4.2.2.1.1. Redes Neuronales Recurrentes (RNN)

Las RNN constituyen una arquitectura de red neuronal en la que se emplean relaciones de recurrencia entre sus elementos. Se ha demostrado su versatilidad en problemas donde se requiere realizar predicciones a partir de una secuencia de *inputs*. Un ejemplo de tales secuencias puede ser una serie temporal, en la cual se recopilan datos y se ordenan cronológicamente. A continuación se muestra el esquema básico de una red recurrente:

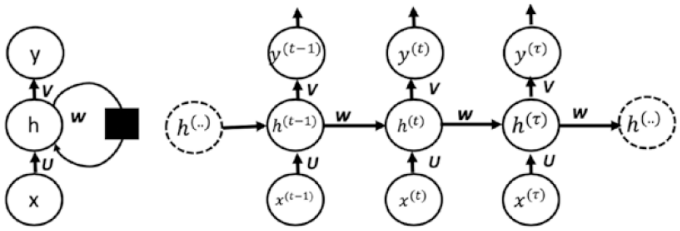

Figura 4.11: Vista esquemática de una RNN
Fuente: Ketkar y Moolayil (2021)

Del lado izquierdo de la figura se observa la característica fundamental de una red neuronal recurrente: existen ciclos en las neuronas. Por otro lado, el lado derecho muestra este ciclo desenvuelto, haciendo evidente la recurrencia al mostrar que el *estado oculto* $h^{(t)}$ se calcula con $h^{(t-1)}$. El *input* de la red de la figura, representado por la secuencia $x^{(1)}, x^{(2)}, ..., x^{(\tau)}$, tiene longitud τ, al igual que el *output*, contenido en la secuencia $y^{(1)}, y^{(2)}, ..., y^{(\tau)}$. Los elementos del *input* se utilizan para apoyar en el cálculo de los estados ocultos de la red. Estos estados ocultos por su parte se emplean para calcular el *output* (Ketkar y Moolayil, 2021).

Un aspecto importante de la figura son las letras que acompañan las conexiones del lado derecho: U, W, V. Estos son los pesos que esta red recurrente emplea para sus predicciones. Naturalmente, el valor de cada uno se determina mediante un proceso de entrenamiento. Es de notar que dicho valor es constante a través de la red. Cabe mencionar que aunque no se hace explícito en la figura, existen sesgos que también es necesario entrenar para el buen funcionamiento del modelo de aprendizaje.

Las RNN pueden clasificarse dependiendo de la longitud de su *input* y *output*. Se tienen las redes *many-to-many*, donde la secuencia de *outputs* tiene múltiples elementos (no necesariamente tendrá la misma longitud que el *input*); y las redes *many-to-one*, donde el *output* lo constituye un solo elemento. Se profundizará en esta última categoría.

Para la red *many-to-one*, considere que el *input* viene dado por la secuencia $x^{(1)}, x^{(2)}, ..., x^{(\tau)}$, y los estados ocultos son $h^{(0)}, h^{(1)}, h^{(2)}, ..., h^{(\tau)}$. Nótese el estado oculto $h^{(0)}$, este es uno de los parámetros de la red que también es necesario entrenar. El cálculo del t-ésimo estado oculto se logra mediante la relación:

$$h^{(t)} = \tanh\left(Ux^{(t)} + Wh^{(t-1)} + b\right) \tag{4.80}$$

donde U y W son pesos y b es un sesgo. La tangente hiperbólica juega el papel de función de activación. Se observa entonces la dependencia del estado oculto anterior y del elemento correspondiente de la secuencia del *input*. Por otro lado, el *output* de la red se calcula con:

$$y = \text{softmax}\left(Vh^{(\tau)} + c\right) \tag{4.81}$$

donde de nuevo, V y c representan un peso y un sesgo, respectivamente. En este caso, softmax es la función de activación.

El entrenamiento de la RNN se logra "desenrollando" la red, esto es, eliminar la recurrencia al escribir todo en términos de $h^{(0)}$. Al desenrollar la RNN, lo que se obtiene es básicamente una red neuronal no recurrente. Esto implica que es posible entrenarla mediante el algoritmo de descenso de gradiente estocástico. El esquema de una red *many-to-one* desenrollada con $\tau = 4$ se muestra a continuación:

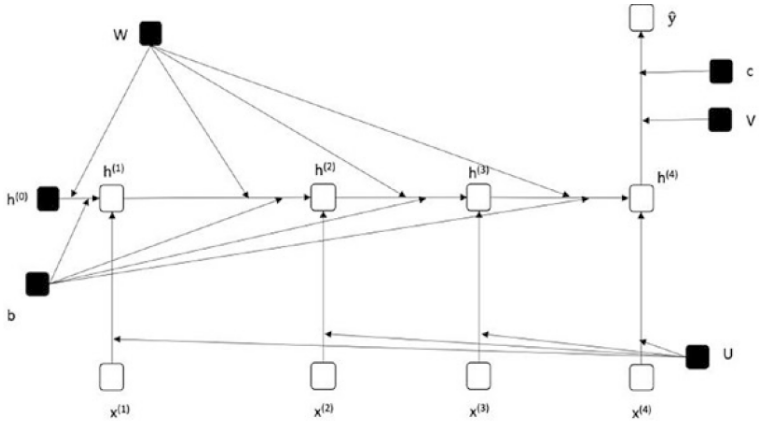

Figura 4.12: RNN *many-to-one* desenrollada
Fuente: Ketkar y Moolayil (2021)

Los cuadros negros representan los parámetros de la red. Se observa que una vez se desenrolla la red desaparecen los ciclos.

Aunque en principio se cuenta con una manera de entrenar la red, las RNN sufren del *problema de desvanecimiento de gradiente*. Este consiste en que las derivadas parciales de la función de costo tienden a cero. Esto implica una desaceleración en el aprendizaje ya que el costo variará bien poco y por ende será difícil minimizarlo. Similarmente, puede que la red sufra del *problema de gradiente explosivo*. Este es el caso contrario, en que las derivadas parciales llegan a tener valores muy grandes. En términos simples, esto surge porque se multiplica repetitivamente una misma matriz durante el cálculo del gradiente. Esto es análogo a que un número en el intervalo $(0, 1)$ tiende a cero cuando se le eleva a una potencia bien alta, o bien, cuando un número mayor a 1 con exponente grande tiende a infinito.

La manera más directa de lidiar con estos problemas es limitar los posibles valores que las derivadas parciales pueden tomar. Sin embargo, esto introduciría un nuevo parámetro que habría que tomar en cuenta a la hora de entrenar la red. Actualmente, una de las alternativas más eficientes para evitar los problemas de desvanecimiento de gradiente y gradiente explosivo es la *memoria larga a corto plazo* o LSTM por sus siglas en inglés.

La LSTM es una variación de las RNN donde los estados ocultos han sido sustituidos por *celdas de memoria*. Estas celdas están constituidas por *puertas*, específicamente la puerta de *input*, la puerta de *output* y la puerta de olvido. Las puertas regulan la información que entra a las celdas y permite que la red "recuerde" datos que van más allá que el último paso recursivo, como se da en una RNN común.

Intuitivamente, dentro de las celdas, la puerta de olvido causa que la información recordada

decaiga i.e se olvide. Si esta puerta no está causando decaimiento, y las puertas de *input* y *output* no están activas, sucede que la celda mantiene su valor o memoria constante. En esta situación, el gradiente también permanecerá invariante. De este modo es que las unidades de LSTM logran evadir problemas que sufren las RNN. Gracias a esto, las LSTM han demostrado ser notablemente superiores en problemas secuenciales, hallando interdependencias complejas en la data (Salehinejad, Sankar, Barfett, Colak, y Valaee, 2017).

4.2.2.1.2. Redes Neuronales Convolucionales (CNN)

Las CNN son una variante de redes neuronales que alcanzan una mejor capacidad predictiva y que pueden ser entrenadas con mayor velocidad (Nielsen, 2015). Esto lo logran basándose en tres principios: *campos receptivos locales*, *pesos compartidos* y *pooling* o *reducción*. El desarrollo de estos conceptos puede facilitarse considerando el ejemplo de reconocimiento de imágenes, donde a partir de la información de sus píxeles, se construye el *input* que se ve en la Figura 4.13.

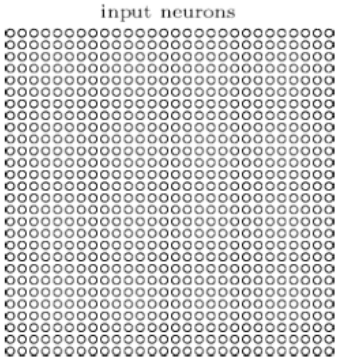

Figura 4.13: *Input* de una CNN. Cada neurona representa un píxel
Fuente: Nielsen (2015)

Los campos receptivos locales surgen al considerar la primera capa oculta de esta red. Suponga que se tiene una neurona de esta capa. En las redes neuronales básicas, cada neurona de la capa de *inputs* tiene una conexión con esta neurona de la capa oculta (Véase Figura 4.10). En una CNN, en cambio, se delimita una región del *input*, denominada campo receptivo local, que será conectada a la neurona. Intuitivamente, se desea que la red identifique una característica específica en esa región. En el ejemplo de reconocimiento de imágenes, esta neurona de la capa oculta estaría analizando exclusivamente una pequeña área de la imagen como se ve en la Figura 4.14.

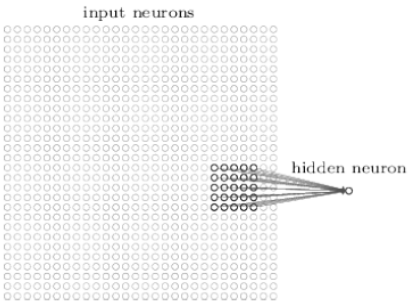

Figura 4.14: Campo receptivo local de una neurona oculta
Fuente: Nielsen (2015)

Naturalmente, a cada conexión de la neurona oculta le corresponde un peso, y dicha neurona cuenta con su propio sesgo. Para analizar todo el *input*, se necesitan múltiples neuronas ocultas, cada una con su campo receptivo local. Así se construye la capa oculta de esta red.

Como se mencionó, cada neurona oculta cuenta con su sesgo y un peso por conexión. Sin embargo, estos pesos y sesgo son compartidos por todas las neuronas de la capa. Es esto a lo que se refiere el principio de pesos compartidos. Dado que todas las neuronas tienen los mismos parámetros, puede interpretarse que buscan la misma característica en la imagen, cada una en su propio campo receptivo local. Al mapeo que va de la capa de *input* a la oculta y que es realizado por el conjunto de parámetros compartidos se le llama *feature map*. Los pesos y sesgo compartidos se llaman en conjunto el *Kernel* del feature map.

Un *feature map* puede identificar una sola característica. Para más características, se emplean múltiples *feature maps*. Este conjunto de feature maps constituye la *capa convolucional* de la red.

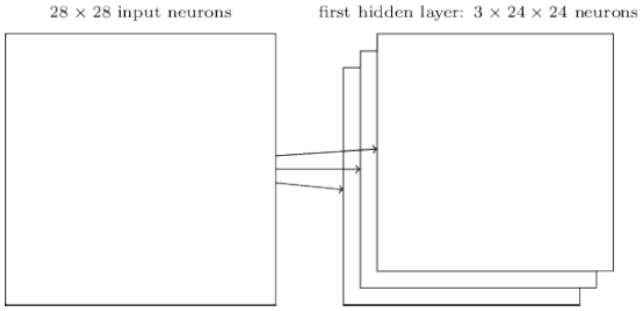

Figura 4.15: Capa de *inputs* y capa convolucional
Fuente: Nielsen (2015)

Acá es posible observar el motivo por el cual las CNN son más fáciles de entrenar: cuentan con significativamente menos parámetros que una red neuronal usual. Esto brinda una ventaja adicional, y es que con menos parámetros se logra un modelo más simple, menos susceptible a sufrir de *overfit*.

El último principio fundamental de las CNN son las capas de *pooling*. Estas se colocan luego de una capa convolucional con el fin de simplificar su *output*. Para esto, una neurona de la capa de *pooling* resume la información depositada en la activación de un grupo de neuronas de la capa convolucional.

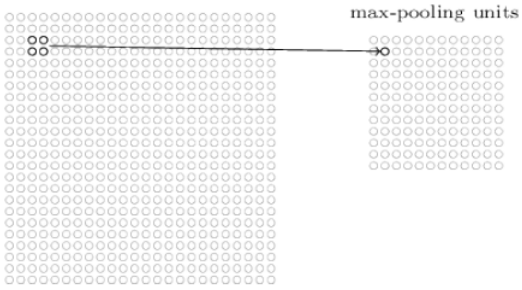

Figura 4.16: Capa de *pooling* resumiendo a capa convolucional
Fuente: Nielsen (2015)

Existen múltiples maneras de llevar a cabo el *pooling*. Por ejemplo, en el *max-pooling* la neurona de la capa de *pooling* toma el valor de activación más alto de su región asignada de la capa convolucional. Como muestra la Figura 4.17, a cada feature map se le hace el procedimiento de *pooling* para condensar la información que contiene.

Finalmente, si se agrega una capa de *outputs* a la red actual, se termina la construcción de una CNN.

Hay varios aspectos que vale la pena resaltar. En primer lugar, el algoritmo de entrenamiento

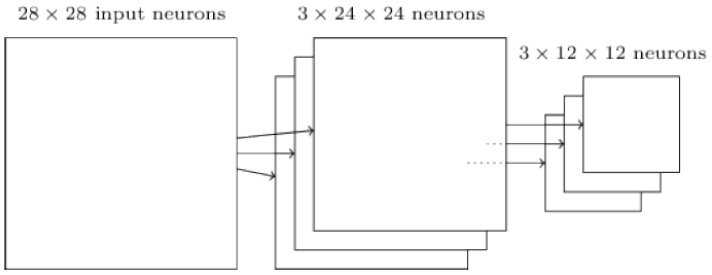

Figura 4.17: CNN con capas de *pooling* agregadas
Fuente: Nielsen (2015)

de esta red sigue siendo descenso de gradiente estocástico. Como se mencionó, esta red es fácil de entrenar debido a la cantidad pequeña de parámetros, en relación a una red neuronal normal. Es gracias a esto también que las CNN por lo general no sufren tantos problemas de gradiente como una RNN. Por otro lado, la arquitectura de las CNN no se limita solo a lo que se ha elaborado. Es posible agregar capas ocultas tradicionales a la red, o bien, más capas convolucionales con sus respectivas capas de *pooling*. Esto puede lograr un mejor desempeño de la red. Similarmente, es posible fusionar una CNN con una RNN, obteniendo así una *red neuronal recurrente convolucional*. Este tema se desarrolla a continuación en la siguiente sección.

4.2.2.1.3. Redes Neuronales Recurrentes Convolucionales (RCNN)

Las RCNN combinan aspectos de las RNN y las CNN. Su arquitectura puede consistir en que el *output* de una CNN se utiliza como el *input* de una RNN (Donahue et al., 2015), o bien, agregando recurrencia a las capas convolucionales de la red (Liang y Hu, 2015). Un ejemplo del primer tipo de RCNN es el que emplea Truong et al. (2018) para el análisis de secuencias de números aleatorios:

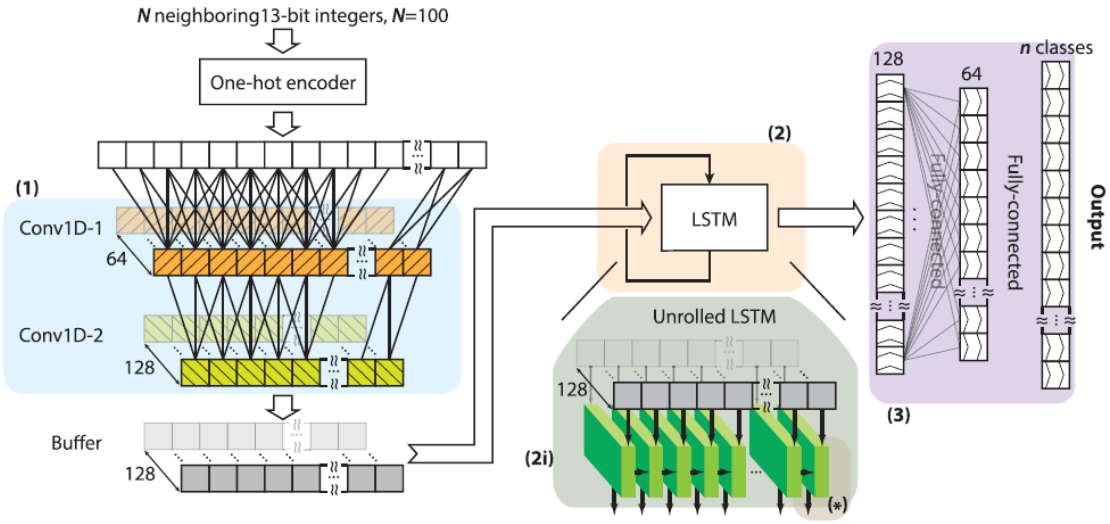

Figura 4.18: Ejemplo de RCNN donde CNN alimenta a RNN
Fuente: Truong et al. (2018)

De la figura, se observa en la sección (1) las capas convolucionales de la red. Estas alimentan a una unidad de LSTM en (2). Finalmente, el *output* de la RNN se inserta en dos capas de neuronas completamente conectadas que producen el *output* global de la red en (3).

4.2.2.2. Support Vector Machine (SVM)

Otro modelo de machine learning empleado en estudios de criptoanálisis es el SVM. La versión más simple de un SVM es el *SVM lineal*. Considere un conjunto de data en el cual cada instancia cuenta con dos características o *features*. Una *feature* puede ser por ejemplo, el tamaño de una persona, o bien su peso. Son características que describen a un punto de la data. Asuma además que cada instancia puede pertenecer a una de dos categorías. Lo que hace el SVM lineal es, a partir del conjunto de entrenamiento, definir una frontera que divide el espacio de *features* de tal modo que de un lado se encuentren las instancias que pertenecen a la categoría 1, y del otro aquellas que forman parte de la categoría 2. Seguidamente esta frontera se emplea para clasificar data nueva.

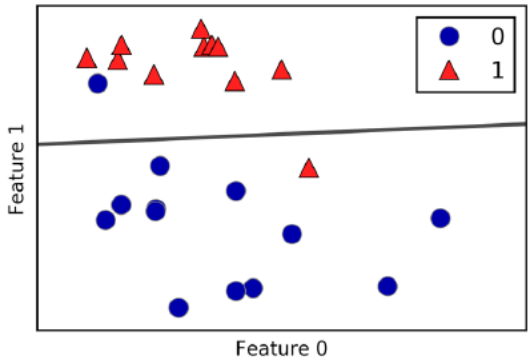

Figura 4.19: Ejemplo de SVM lineal
Fuente: Müller y Guido (2016)

La Figura 4.19 muestra un ejemplo de SVM lineal. Las categorías son representadas por los círculos azules y triángulos rojos. Se observan puntos atípicos de entrenamiento mal clasificados los cuales, en principio, pueden clasificarse bien forzando al SVM a construir una frontera distinta. Sin embargo, al ajustarse a puntos atípicos es probable que el modelo tendrá problemas de *overfit*. Un aspecto a resaltar es que la frontera es una recta, de acá el nombre SVM lineal. Si se tuviera un espacio de 3 *features*, la frontera sería un plano, y para más dimensiones sería un hiperplano.

El SVM lineal tiene un buen desempeño cuando las regiones que ocupan las categorías en el espacio de *features* pueden separarse fácilmente mediante un modelo lineal. Sin embargo, considere la situación de la Figura 4.20. Por más que se intente, es imposible definir una recta que divida satisfactoriamente la data. En este caso, se hace uso del *kernelized SVM* o solamente SVM.

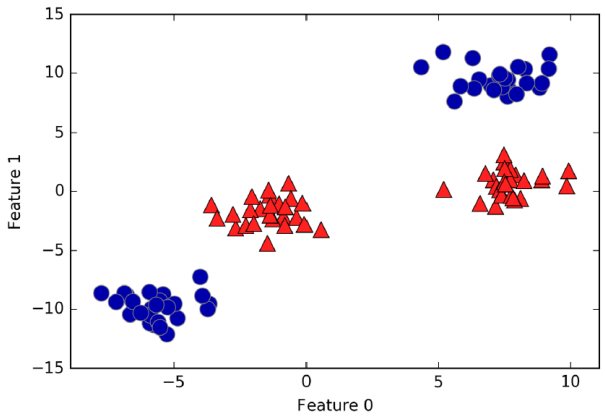

Figura 4.20: Data que no puede ser separada por un SVM lineal
Fuente: Müller y Guido (2016)

Un SVM es la generalización del caso lineal que permite lidiar con problemas como los de la Figura 4.20. Para ello, se agregan *features* no lineales. Por ejemplo, en la Figura 4.21 se tiene el mismo espacio de *features* pero se agrega una nueva dimensión dada por feature0^2. En el nuevo espacio es posible ahora llevar a cabo una separación lineal con un plano. En la representación bidimensional original, la frontera ya no será dada por un modelo lineal como se observa en la Figura 4.22.

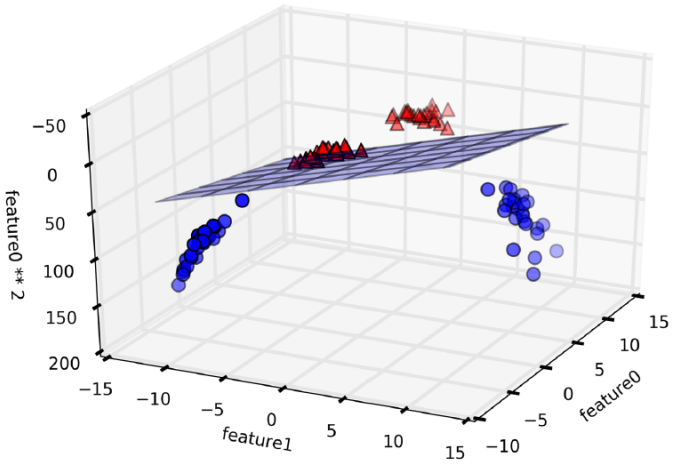

Figura 4.21: Espacio de *features* expandido con plano de frontera
Fuente: Müller y Guido (2016)

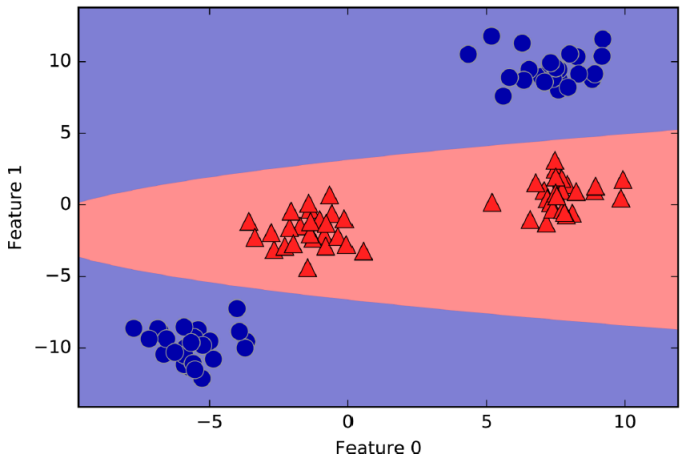

Figura 4.22: Frontera no lineal en espacio bidimensional
Fuente: Müller y Guido (2016)

El SVM lo que logra entonces es realizar una *transformación* del espacio de *features*. En el nuevo espacio, podrá aplicarse una frontera lineal con la cual se resolverá el problema de clasificación. El proceso que se sigue para determinar la transformación correcta no se cubre en el presente trabajo, pero puede consultarse Müller y Guido (2016) para más información.

4.2.3. Machine Learning aplicado a generación de números aleatorios

Como ya sugerido en la sección de redes neuronales recurrentes convolucionales, una clase de estudio criptoanalítico que es posible llevar a cabo con machine learning es el análisis de secuencias

de números aleatorios. El proceso de criptoanálisis consiste en que el modelo de machine learning analiza el *output* de un RNG e intenta identificar patrones para que dada una secuencia de números aleatorios, sea capaz de predecir el siguiente con una versatilidad mayor a la que se tiene solamente adivinando.

En un ataque de machine learning a un PRNG, sería de esperarse que un modelo de aprendizaje avanzado tendrá la capacidad de predecir los *outputs* del generador sin importar lo robusto que este sea. En cambio, podría parecer redundante aplicar un modelo de machine learning a un TRNG, dado que este es, en teoría, completamente aleatorio. Sin embargo, en el proceso de extracción de aleatoriedad pueden existir imperfecciones físicas o bien fuentes de ruido externas capaces de producir patrones y comprometer la seguridad del TRNG (Li et al., 2020). Por ello, es necesario certificar la calidad de un TRNG mediante técnicas confiables.

Se han realizado ataques tanto a PRNGs como TRNGs con resultados variados. Feng y Hao (2020), emplearon una red neuronal prealimentada con cuatro capas para atacar al generador lineal congruencial (LCG), Mersenne Twister (MT) y un QRNG. LCG y MT son PRNGs. Los resultados mostraron que el modelo de aprendizaje fue capaz de superar la seguridad del LCG, pero no logró predicciones satisfactorias sobre MT y el QRNG. Es de notar que el modelo falló con un PRNG debido a su relativa simpleza. Fan y Wang (2018) hicieron uso de una arquitectura de red similar para predecir satisfactoriamente una secuencia pseudoaleatoria basada en el número π. Por otro lado, Kim et al. (2017) analizaron un TRNG de ruido telegráfico aleatorio aplicando una RNN con arquitectura de LSTM. Los resultados indicaron que el generador no era predecible. Un estudio por Chai et al. (2019) obtuvo resultados similares al analizar un TRNG mediante una LSTM. Por su parte, Li et al. (2020) generaron una LSTM con *temporal pattern attention* (TPA) para atacar un TRNG y detectaron patrones en ciertas fases del proceso de extracción de aleatoriedad. No obstante, el *output* final del TRNG estuvo libre de patrones. Cabe mencionar el trabajo realizado por Truong et al. (2018) sobre un QRNG. En este caso se empleó una RCNN que validó la aleatoridad del *output* posprocesado del generador al ser incapaz de realizar predicciones acertadas sobre este.

CAPÍTULO 5

Metodología

Motivado por los antecedentes mencionados al final de la sección 4.2.3, el análisis del QRNG se llevó a cabo empleando tres distintos modelos de aprendizaje supervisado, específicamente, *la red neuronal prealimentada* (FNN), la *red neuronal recurrente con memoria larga a corto plazo* (LSTM) y la *red neuronal recurrente convolucional* (RCNN), esto con el fin de comparar el desempeño de las tres técnicas. Las implementaciones se realizaron en el lenguaje Python, específicamente, se hizo uso de la API de Deep Learning *keras*, que permite la elaboración eficaz de modelos de redes neuronales.

Adicional al estudio del QRNG, y con el propósito de demostrar las capacidades de predicción de los modelos de aprendizaje sobre secuencias numéricas que cuentan con correlaciones, se desarrolló un ataque sobre el PRNG de congruencia lineal (LCG) descrito en la sección 4.1.1.

Este capítulo sigue la siguiente estructura: primero se caracteriza la data y se detalla la preparación que se le dio para ser utilizada en el entrenamiento de las redes. A esto le sigue la descripción de cada modelo de machine learning, haciendo énfasis en las arquitecturas de red empleadas tanto en el problema del PRNG como del QRNG. Finalmente, se abarcan generalidades de la fase de entrenamiento y se mencionan aspectos relevantes sobre la manera en la que se recolectaron e interpretaron los resultados finales.

5.1. Caracterización y preparación de la data

En el contexto del QRNG se contó con dos conjuntos de data: la data *posprocesada*, que consiste de dígitos binarios; y la *cruda*, constituida por números de 16 bits que siguen una distribución gaussiana. Se llevaron a cabo estudios sobre cada conjunto de manera independiente a fin de observar si contaban con patrones subyacentes en sus secuencias. La data cruda en particular se sometió a un proceso de *binning* como el descrito en la sección 4.1.4.4.1 con el fin de obtener una distribución uniforme de números de 4 bits (ver Figuras 5.1 y 5.2), ya que esto redujo significativamente el costo computacional de la implementación. En efecto, si se utilizan números de 16 bits, solamente la capa de *outputs* de todos los modelos requeriría de $2^{16} = 65536$ neuronas, una por cada posible número producido por el generador, mientras que con números de 4 bits esto se reduce a solamente 16 neuronas. De las consideraciones anteriores, para la data cruda se tuvo una probabilidad de adivinar $P_A = \frac{1}{16} = 0.0625$, mientras que para la posprocesada $P_A = \frac{1}{2} = 0.50$.

Dado que se utilizó el enfoque de aprendizaje supervisado, por cada muestra de entrenamiento

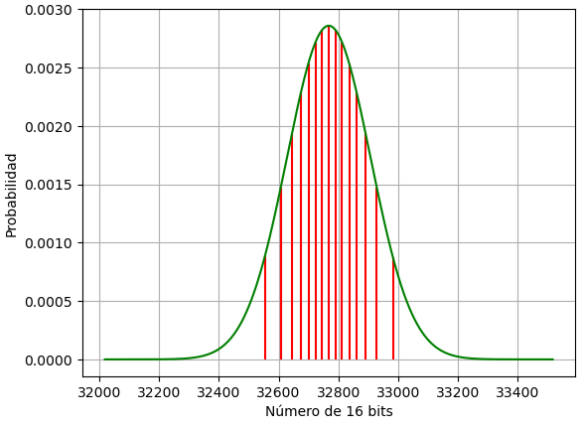

Figura 5.1: División de distribución gaussiana en intervalos equiprobables
Fuente: elaboración propia

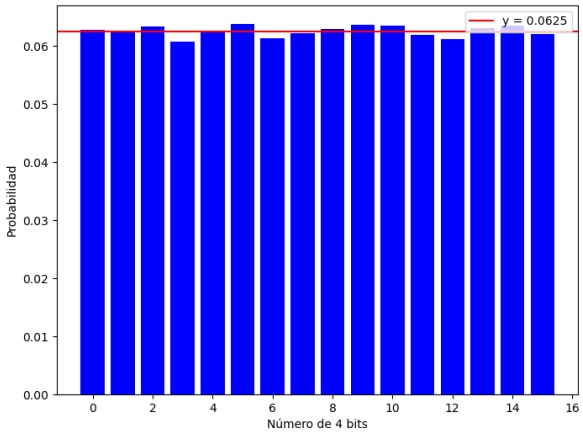

Figura 5.2: Distribución uniforme obtenida luego de proceso de *binning*
Fuente: elaboración propia

fue necesario proveerle a la red un *input* con su *output* respectivo. Con el fin de observar si la red era capaz de predecir el siguiente número de la secuencia (o cadena) con una probabilidad mayor a P_A partiendo de las cifras previas, se definió la *longitud de input* L y el *Salto* o *Corrimiento* S. El *input* de una muestra de entrenamiento se construyó tomando L números consecutivos de la secuencia aleatoria siendo su *output* el número $L+1$. Luego, la siguiente muestra se obtuvo realizando un corrimiento de S números en la secuencia y repitiendo el proceso de tomar los primeros L números como *input* y el $L+1$ como *output*. Por ejemplo, sea $a_1, a_2, a_3...$ la cadena producida por el generador. Para la primera muestra, el *input* será $a_1, a_2, ..., a_L$ con *output* a_{L+1}. La segunda muestra tendrá entonces como *input* $a_{S+1}, a_{S+2}, ..., a_{L+S}$ con a_{L+S+1} como *output* correspondiente. Así, este proceso de corrimiento se repitió hasta haber recorrido toda la salida del generador.

Para los conjuntos de entrenamiento y prueba se empleó una cadena de longitud 5 millones tanto en el caso de la data cruda como posprocesada. Un 50 % se utilizó con el fin de entrenar, y el resto sirvió como la data de prueba. Por otro lado, se tomó un 10 % de la data de entrenamiento con el propósito de construir el conjunto de validación. El valor de L en un inicio se estableció como 25, pero se varió durante distintas sesiones de entrenamiento con la intención de medir su impacto sobre el desempeño de las redes. Por su parte, se tuvo $S = 1$ siempre.

En lo que concierne al ataque sobre el LCG, los parámetros del generador empleado fueron

$a = 25214903917$, $c = 1$ y $\mathcal{M} = 2^{20}$. Las cifras producidas se convirtieron a números de 8 bits, y las muestras de entrenamiento se prepararon del mismo modo que en el caso del QRNG. El conjunto de entrenamiento contó con una longitud del 80 % del periodo $\mathcal{M}$ y el conjunto de prueba fue el 20 % sobrante. Se empleó en todo caso $L = 13$ y $S = 1$. Dado que la distribución de este PRNG es uniforme (ver Figura 4.1), la probabilidad de adivinar fue $\frac{1}{2^8} \approx 0.0039$.

5.2. Red neuronal prealimentada

El primer modelo a considerar es la FNN. Su capa de *inputs* consistió de L neuronas, de tal modo que en cada una se almacenaba un número de la cadena que sirve como *input* de la muestra de entrenamiento. Tanto para el LCG, así como para la data cruda del QRNG, se llevó a cabo un proceso de normalización, dividiendo los números de la entrada entre 256 y 16, respectivamente, tal que todos los miembros del *input* pertenecieran al intervalo $[0, 1)$. Intuitivamente, esto ayuda a reducir la complejidad durante el aprendizaje dado que evita el manejo de números grandes.

A continuación se agregaron dos capas ocultas, las cuales, en el análisis del LCG de 8 bits, poseyeron $2^8 = 256$ neuronas cada una, con función de activación ReLU de acuerdo a lo propuesto por Li et al. (2020). A partir de esto, se tomó como punto de partida en el estudio del QRNG emplear capas ocultas de 2^4 neuronas tanto para la data cruda como la posprocesada (heurísticamente, detectar patrones en la data posprocesada es al menos tan difícil como hallarlos en la data cruda). La función de activación de dichas capas fue ReLU, debido a que su rango no se ve limitado como las funciones sigmoide o tangente hiperbólica. Se estudió si aumentar el número de neuronas ocultas para distintas sesiones de entrenamiento producía mejores resultados, o si solamente llevaba a redes que sufren de *overfit* respecto de la data de entrenamiento.

A la salida de cada capa oculta se aplicó el proceso de *Normalización por lotes*, el cual acelera la fase de entrenamiento, y reduce (aunque no elimina) la necesidad de utilizar *Dropout* (Ioffe y Szegedy, 2015).

Finalmente, se añadió la capa de *outputs* de la red. En el caso del LCG, esta poseyó 256 neuronas debido a que este PRNG produce números enteros de 8 bits, i.e. pertenecen al intervalo $[0, 255]$. Se utilizó softmax como función de activación. De manera similar, la red de la data cruda contó con 16 neuronas en su capa de *outputs* con activación softmax. En lo que concierne a la data posprocesada, la capa de *outputs* solamente tuvo una neurona con activación sigmoide. El motivo de esta diferencia se detalla en la sección 5.5.

5.3. Red neuronal recurrente con arquitectura LSTM

El modelo de red recurrente poseyó como primer elemento una capa LSTM que tuvo la funcionalidad adicional de ser la capa de *inputs*. A diferencia de la FNN, fue necesario codificar las entradas con vectores *one-hot*. Un vector *one-hot* es aquel cuyas componentes son todas cero a excepción de una sola, cuyo valor es uno. Su dimensión corresponde al número de elementos distintos que se desea codificar. Por ejemplo, el conjunto $\{0, 1, 2, ..., n\}$ se expresa en términos de vectores *one-hot* de acuerdo con el siguiente mapeo:

$$0 \to \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad 1 \to \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad 2 \to \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad \cdots \quad n \to \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \tag{5.1}$$

Así, si el primer *input* de entrenamiento es la secuencia 2 0 1..., la red LSTM se alimentaría con el elemento matricial

$$\left[\begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \cdots \right] \tag{5.2}$$

Ahora bien, el número de unidades (que definen la dimensión del *output* de esta capa) que tuvo la capa LSTM fue de 128 en el análisis del LCG. Empleando argumentos similares al caso de la FNN, se utilizó como punto de partida en ambos conjuntos de datos del QRNG un total de unidades en el orden de 16, específicamente 32. Se realizaron igualmente variaciones a este número en distintas sesiones de entrenamiento con el fin de observar su impacto sobre el desempeño de la red. La función de activación en todo caso fue ReLU. Seguidamente se agregó una capa oculta tradicional de 128 neuronas en el caso del LCG, y 32 neuronas, en un inicio, para el QRNG. De nuevo, se varió este número con el propósito de medir su efecto sobre las predicciones de la red. Para ambos generadores, esta capa contó con activación ReLU. Cabe mencionar que tanto la capa LSTM como la oculta se sometieron al proceso de normalización por lotes. Para finalizar se insertó la capa de *outputs* con las mismas características para cada caso descritas en la sección anterior.

5.4. Red neuronal recurrente convolucional

En lo que concierne a la RCNN, la arquitectura consistió de una capa convolucional seguida de una LSTM. Según Truong et al. (2018), la capa convolucional tiene el papel de extraer características (o *features*) de la data para que la LSTM aprenda todo el *contexto* clásico, i.e. ruido electrónico, térmico, etc. Similar al caso de la red recurrente con LSTM, los *inputs* de la red se codificaron mediante vectores *one-hot*.

En un inicio, para el LCG, la red contó con una capa convolucional de 64 filtros, que son los *feature maps* descritos en la sección 4.2.2.1.2; con un *max-pooling* de tamaño 2 y activación ReLU. Seguidamente se agregó una capa LSTM de 128 unidades con tangente hiperbólica como función de activación. Estos parámetros estuvieron de acuerdo con los empleados por Truong et al. (2018), pero a diferencia de ellos, se optó por utilizar una sola capa convolucional en lugar de 2 porque se trabajó con números de 8 bits o menos, lo cual constituye un problema más simple comparado a las secuencias de números de 13 bits analizadas en el mencionado estudio. Por último, se añadió una capa oculta tradicional de 64 neuronas con activación ReLU seguida de la capa de *outputs* idéntica a la descrita en la sección 5.2. Es de notar que también se aplicó normalización por lotes a los componentes de esta red.

Por otro lado, tanto para la data cruda como la posprocesada del QRNG se tuvo el mismo estilo de RCNN, pero los filtros de la capa convolucional, las unidades de la LSTM, y el número de neuronas en la capa oculta se inicializaron todos en 32. Nuevamente, se variaron estos parámetros a fin de determinar si una RCNN más complicada puede llevar a mejores predicciones.

5.5. Fase de entrenamiento

Al terminar la construcción de las redes se procedió a entrenarlas. Naturalmente, los aspectos relevantes de este proceso son la función de costo, las técnicas de regularización utilizadas para mitigar el *overfit*, y los hiperparámetros de las redes, entre otros.

En lo que concierne a la función de costo, se utilizó "*categorigal crossentropy*" tanto para el LCG como la data cruda. De acuerdo con keras, esto implicó que los *outputs* de las muestras se codificaran en vectores *one-hot*. Por otro lado, en el caso de la data posprocesada la función de costo fue "*binary crossentropy*". Este es el caso binario de la función de entropía cruzada que ofrece keras para problemas de clasificación binaria. Su uso requiere que la capa de *outputs* de la red esté constituida de una sola neurona con función de activación sigmoide, como se describió anteriormente. Nótese que acá no se necesitó llevar a cabo una codificación *one-hot* para los *outputs*.

En principio no se aplicó ninguna técnica de regularización a las redes. Sin embargo, en casos donde estas mostraban claras señales de *overfit*, se procedió a emplear *Dropout* y regularización L2 donde fuese pertinente.

Por otro lado, la tasa de aprendizaje se estableció como 0.0005 para los datos del QRNG y 0.001 en el caso del LCG, lo cual aseguró una evolución estable de la red. El tamaño de los mini-lotes se fijó en 1024, esto con el fin de obtener mejores estimaciones del gradiente en cada mini-lote de acuerdo con lo indicado en la ecuación (4.75). El número máximo de *epochs* fue 50 (200 para el LCG), y en cada uno se evaluó la precisión de las redes sobre el conjunto de validación con intención de observar su desempeño sobre data que no fue utilizada con el fin de entrenarlas. Esto se logró obteniendo el *costo de validación* y la *precisión de validación* por *epoch*, que son la función de costo evaluada sobre el conjunto de validación y el porcentaje de aciertos que tiene la red sobre este mismo conjunto, respectivamente. Considerando que un costo más pequeño implica una red con mejor desempeño, el entrenamiento procedía a detenerse si en 10 *epochs* consecutivos no se lograba reducir más el costo de validación, esto con el propósito de evitar problemas de *overfit*. Terminado el entrenamiento, se guardó la red cuyos parámetros obtuvieron el menor costo de validación.

5.6. Recolección e interpretación de resultados

El resultado principal de cada red fue su desempeño sobre el conjunto de prueba. En el análisis de los datos asociados al QRNG se incluyeron además los parámetros de las sesiones de entrenamiento de cada red como número de neuronas, valor de L, etc., así como notas acerca de las medidas que se tomaron a fin de mitigar el *overfit* en caso que fuesen necesarias. Con esta información, para cada tipo de red se eligió la mejor instancia basándose en dos criterios: la precisión sobre el conjunto de prueba y la simplicidad del modelo. Este último criterio significa que si al aumentar la complejidad de una red esta no logra mejores resultados, se conserva su versión más simple. Adicionalmente, se construyeron gráficas de precisión contra *epoch* y costo contra *epoch* para cada una de las redes que se consideraron la mejor de su categoría. Esto con la intención de analizar la evolución de los modelos durante el entrenamiento.

Finalmente, en el contexto de la data cruda así como la posprocesada, los resultados de cada red elegida se sometieron a la prueba t de muestras emparejadas (con significancia $\alpha = 0.001$) a fin de darles validez estadística. (Anderson, Sweeney, Williams, Camm, y Cochran, 2014). Para esto, se tomó un conjunto de 2000000 de datos nuevos y se dividió en 500 subconjuntos de tamaño 4000. Para cada *input* de un subconjunto dado se intentó predecir el *output* correspondiente con 2 métodos: el método A, que consistió en emplear la red neuronal; y el método B, que se basó en adivinar con un número al azar que se encuentre en el rango de posibles valores del generador. Se contó el número de aciertos obtenidos por cada método por subconjunto, así, se obtuvieron 500 datos emparejados que fueron sometidos a la mencionada prueba estadística. Los detalles de este procedimiento se omiten en el trabajo principal, pero pueden consultarse en el Apéndice B.

CAPÍTULO 6

Resultados

6.1. Desempeño de modelos de aprendizaje sobre data cruda del QRNG

A continuación se resumen los resultados obtenidos al analizar la data cruda del generador con los tres tipos de redes neuronales. Los Cuadros 6.1 a 6.3 incluyen los parámetros y el valor de L utilizados en distintas sesiones de entrenamiento de cada red, con sus precisiones respectivas sobre el conjunto de prueba. Nótese que en las tablas el término "CO" significa "capa oculta".

No.	Neuronas CO1	Neuronas CO2	Longitud *input* (L)	Precisión sobre conjunto de prueba	Regularización
1	16	16	25	0.0718	-
2	32	32	25	0.0719	-
3	64	64	25	0.0722	*Dropout*
4	128	128	25	0.0719	*Dropout*
5	256	128	25	0.0715	*Dropout*
6	64	64	32	0.0720	*Dropout*
7	64	64	50	0.0721	*Dropout*
8	64	64	13	0.0721	*Dropout*

Cuadro 6.1: Resultados de FNN respecto la data cruda

No.	Unidades LSTM	Neuronas CO	Longitud *input* (L)	Precisión sobre conjunto de prueba	Regularización
1	32	32	25	0.0714	-
2	64	64	25	0.0718	*Dropout*
3	128	128	25	0.0713	*Dropout*
4	64	64	32	0.0720	*Dropout*
5	64	64	50	0.0721	*Dropout*
6	64	64	13	0.0720	*Dropout*
7	64	64	15	0.0720	*Dropout*

Cuadro 6.2: Resultados de LSTM respecto la data cruda

No.	Filtros CC	Unidades LSTM	Neuronas CO	Longitud *input* (L)	Precisión sobre conjunto de prueba	Regularización
1	32	32	32	25	0.0715	*Dropout*
2	32	64	32	25	0.0716	*Dropout*
3	32	64	64	25	0.0708	*Dropout*
4	64	64	64	25	0.0707	*Dropout*
5	32	64	32	32	0.0719	*Dropout*
6	32	64	32	50	0.0716	*Dropout*
7	32	64	32	13	0.0713	*Dropout*

Cuadro 6.3: Resultados de RCNN respecto la data cruda

Como se observa en los cuadros, primero se variaron los parámetros propios de la red. Una vez se hallaron valores óptimos de estos (es decir, valores de acuerdo al criterio de simplicidad del modelo descrito en la sección 5.6), se procedió a variar L que inicialmente se encuentra fijo en 25.

Basándose en los criterios de precisión sobre el conjunto de prueba y simplicidad del modelo, para la FNN se eligió como mejor instancia la red número 3 del Cuadro 6.1. Similarmente, las redes 4 y 5 se seleccionaron como mejores instancias para la LSTM y la RCNN, respectivamente. Las Figuras 6.1 a 6.3 muestran la evolución de los 3 modelos elegidos durante la fase de entrenamiento en forma de gráficas de precisión contra *epoch* y costo contra *epoch* tanto sobre el conjunto de entrenamiento como el de validación.

Es de notar que en todas las instancias donde se empleó *Dropout* en la red prealimentada, se hizo aplicando la técnica de regularización a ambas capas ocultas, con una fracción del 20 % de las neuronas siendo borradas. Para la red recurrente, el parámetro de *Dropout* se fijó en 10 %, siendo aplicado tanto a la capa LSTM como la oculta. En el caso de la RCNN, el *Dropout* fue siempre del 10 %, utilizado en la capa convolucional y la LSTM, excepto en la primera instancia, donde solamente fue empleado en la capa LSTM con un parámetro del 20 %.

Un aspecto relevante es que la RCNN mostró estar más propensa a sufrir de *overfit* en comparación a los otros dos modelos. En efecto, durante las sesiones de entrenamiento se observó a menudo que aunque la precisión sobre el conjunto de validación ya había convergido, la precisión sobre el conjunto de entrenamiento seguía aumentando.

Por su parte, los resultados de las pruebas estadísticas realizadas sobre las tres redes escogidas proveyeron los valores p que se observan en el Cuadro 6.4.

Modelo	Valor p
FNN	3.031×10^{-142}
LSTM	6.556×10^{-141}
RCNN	1.804×10^{-141}

Cuadro 6.4: Valores p en el caso de la data cruda

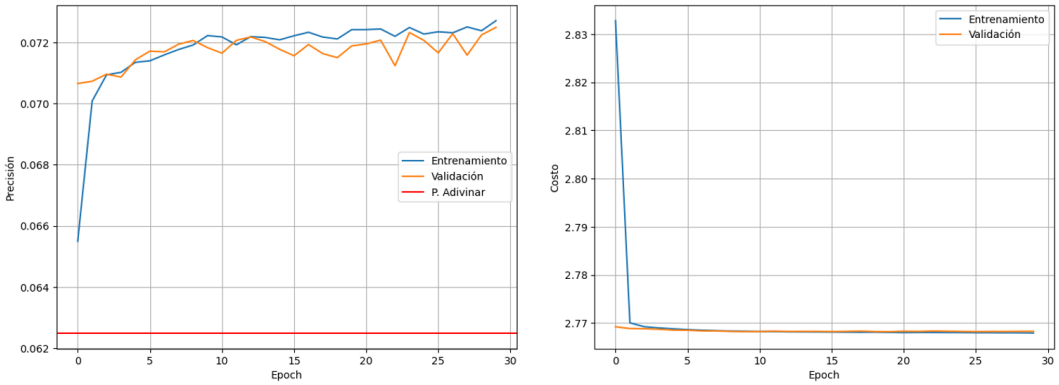

Figura 6.1: Evolución de la FNN en el estudio de la data cruda
Fuente: elaboración propia

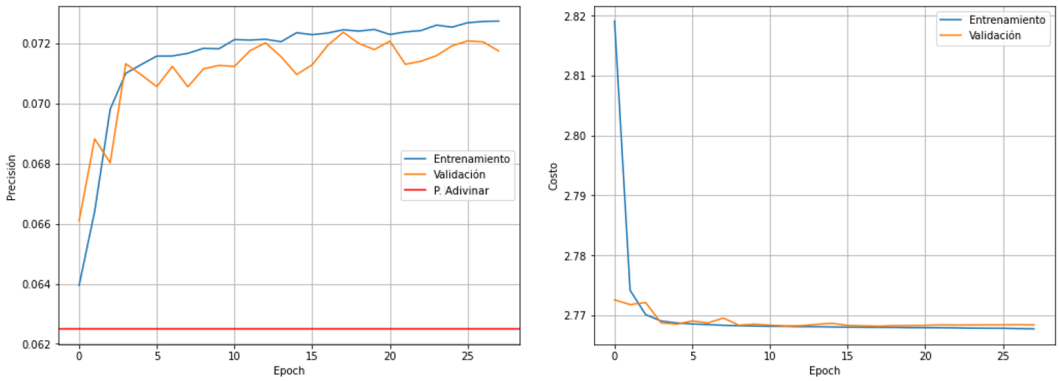

Figura 6.2: Evolución de la LSTM en el estudio de la data cruda
Fuente: elaboración propia

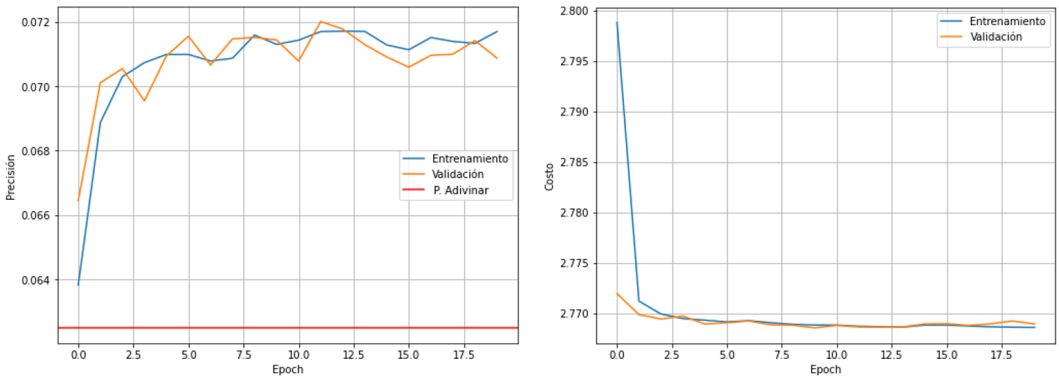

Figura 6.3: Evolución de la RCNN en el estudio de la data cruda
Fuente: elaboración propia

6.2. Desempeño de modelos de aprendizaje sobre data posprocesada del QRNG

Seguidamente se presentan los resultados obtenidos del análisis de la data posprocesada del QRNG. Similar al caso de la data cruda, los Cuadros 6.5 a 6.7 recopilan las distintas sesiones de entrenamiento realizadas con cada modelo. En ningún caso se eligió una mejor instancia ya que no se tuvo una red que hallará patrones subyacentes en la data. Sin embargo en las Figuras 6.4 a 6.6 se muestran las evoluciones de las instancias 5, 4 y 2 de la FNN, LSTM y RCNN, respectivamente; y en el Cuadro 6.8 se adjuntan los valores p obtenidos para estas mismas instancias.

No.	Neuronas CO1	Neuronas CO2	Longitud *input* (L)	Precisión sobre conjunto de prueba	Regularización
1	16	16	25	0.5002	-
2	64	64	25	0.5002	*Dropout*
3	128	64	25	0.4996	*Dropout* y L2
4	128	64	32	0.4996	*Dropout* y L2
5	128	64	50	0.5001	*Dropout* y L2
6	128	64	15	0.4996	*Dropout* y L2
7	128	64	7	0.4996	*Dropout* y L2

Cuadro 6.5: Resultados de FNN respecto la data posprocesada

No.	Unidades LSTM	Neuronas CO	Longitud *input* (L)	Precisión sobre conjunto de prueba	Regularización
1	32	32	25	0.4998	-
2	64	64	25	0.5002	-
3	64	128	25	0.5000	-
4	64	64	32	0.5001	-
5	64	64	50	0.5008	-
6	64	64	13	0.5002	-
7	64	64	15	0.4997	-

Cuadro 6.6: Resultados de LSTM respecto la data posprocesada

No.	Filtros CC	Unidades LSTM	Neuronas CO	Longitud *input* (L)	Precisión sobre conjunto de prueba	Regularización
1	32	32	32	25	0.5005	*Dropout*
2	32	64	32	25	0.5001	*Dropout*
3	32	64	64	25	0.5003	*Dropout*
4	64	64	64	25	0.5001	*Dropout*
5	32	64	32	32	0.5003	*Dropout*
6	32	64	32	50	0.5005	*Dropout*
7	32	64	32	13	0.5003	*Dropout*

Cuadro 6.7: Resultados de RCNN respecto la data posprocesada

Como nota, en el caso de la FNN, tanto el *Dropout* como la regularización L2 se aplicaron a ambas capas ocultas con 20 % de neuronas borradas y $\lambda = 0.0001$, respectivamente. En lo que concierne a la RCNN, el *Dropout* fue solamente empleado en la capa LSTM con un parámetro del 10 %.

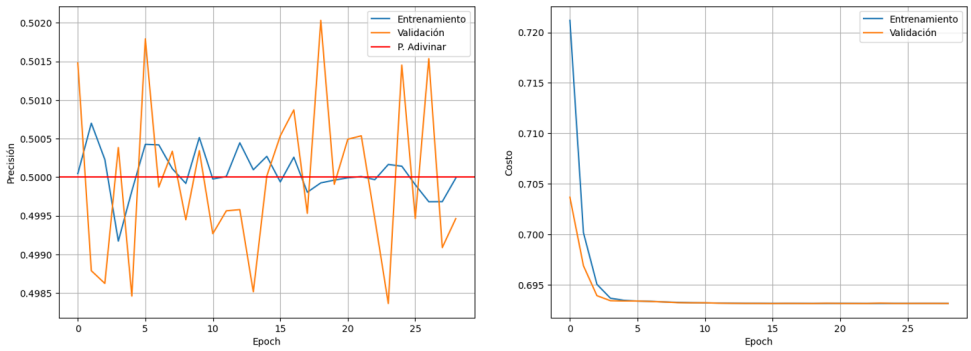

Figura 6.4: Evolución de la FNN en el estudio de la data posprocesada
Fuente: elaboración propia

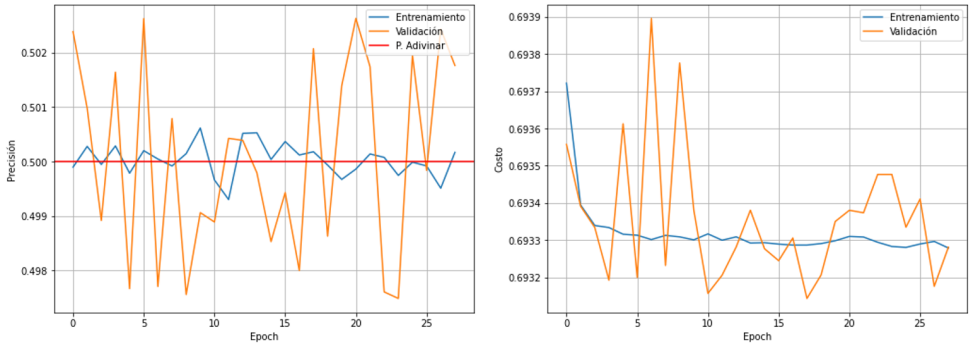

Figura 6.5: Evolución de la LSTM en el estudio de la data posprocesada
Fuente: elaboración propia

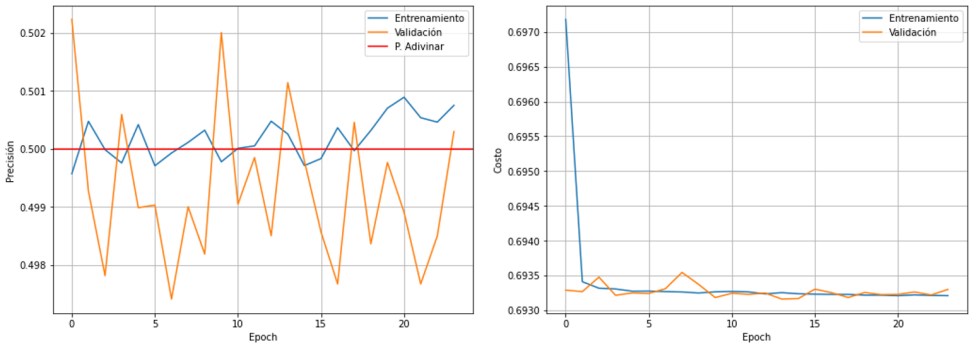

Figura 6.6: Evolución de la RCNN en el estudio de la data posprocesada
Fuente: elaboración propia

Modelo	Valor p
FNN	0.1531
LSTM	0.0735
RCNN	0.1247

Cuadro 6.8: Valores p en el caso de la data posprocesada

6.3. Resultados sobre el PRNG de congruencia lineal

En lo que concierne al análisis del LCG, se lograron precisiones sobre el conjunto de prueba de 0.9733, 0.8846 y 0.8485 con los modelos de FNN, LSTM y RCNN, respectivamente. Estas superan marcadamente la probabilidad de adivinar que en este caso es $P_A = 0.0039$. En las Figuras 6.7 a 6.9 se incluyen gráficas de precisión contra *epoch* y costo contra *epoch*.

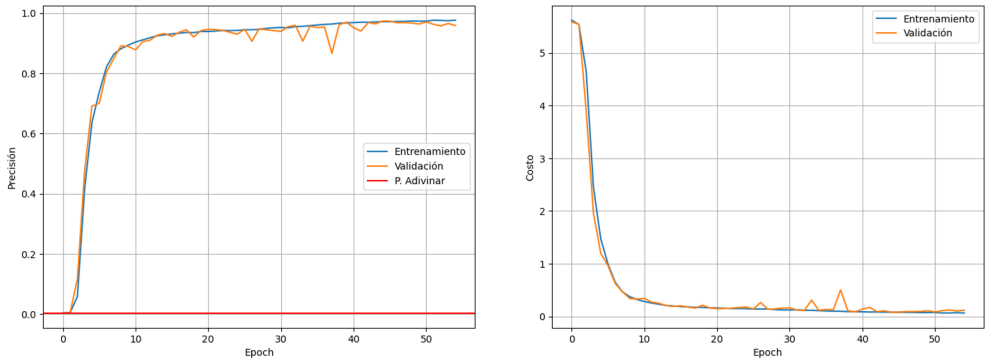

Figura 6.7: Evolución de la FNN en el estudio del LCG
Fuente: elaboración propia

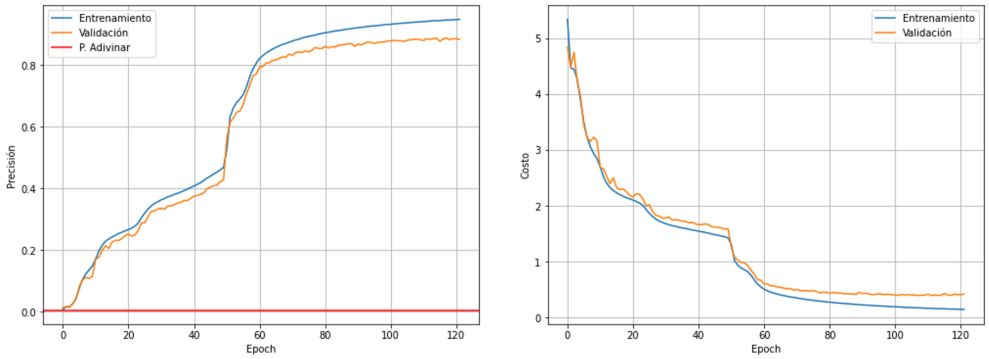

Figura 6.8: Evolución de la LSTM en el estudio del LCG
Fuente: elaboración propia

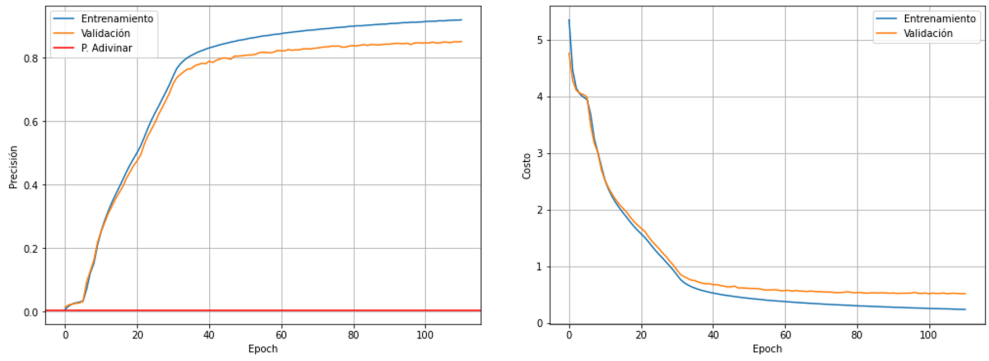

Figura 6.9: Evolución de la RCNN en el estudio del LCG
Fuente: elaboración propia

CAPÍTULO 7

Análisis y discusión de resultados

El estudio de la data cruda reveló que los tres modelos de aprendizaje fueron capaces de detectar patrones intrínsecos en la secuencia de números de 4 bits. En efecto, se obtuvieron precisiones que rondaron alrededor de 0.0720, una mejora de casi 0.01 o 1 % respecto de $P_A = 0.0625$. Adicionalmente, estos resultados fueron estadísticamente significativos debido a que los valores p obtenidos fueron menores que la significancia $\alpha = 0.001$. Considerando el *binning* realizado para preparar la data cruda, los resultados indican que la distribución gaussiana del ADC cuenta con correlaciones en el *orden* que es construida, ya que es posible predecir el bin al que pertenece el siguiente número de la secuencia, con una exactitud ligeramente mayor a la que se tendría solamente adivinando, si se conocen los bins correspondientes de los números anteriores (acá "bin" se refiere a cada uno de los 16 intervalos equiprobables de la distribución gaussiana).

En principio, lo anterior confirma la existencia de patrones dentro de la data cruda del generador que se asocian a ruido clásico. No obstante, el esquema empleado no provee una manera de predecir el siguiente número de 16 bits, solamente reduce las posibilidades al intervalo equiprobable predicho por la red (i.e. el número de 4 bits). Un ataque quizá más efectivo se lograría hallando patrones al aplicar un *binning* con más de 4 bits, dado que esto tiene el efecto de reducir el ancho de los intervalos equiprobables. Sin embargo, en la práctica aumentar el número de bits implica un incremento en la cantidad de recursos computacionales necesarios para realizar un ataque exitoso. En el presente trabajo, se requirieron optimizaciones en el código y se utilizaron recursos externos (ver apéndice A) para lograr un ataque viable sobre el generador considerando solamente 4 bits y una secuencia de 5 millones de números aleatorios para construir los conjuntos de entrenamiento y prueba. Aumentar el número de bits supondría una disminución en el tamaño de la secuencia para compensar el uso de recursos, y por ende se tendría un entrenamiento de menor calidad. Esto es especialmente cierto para la LSTM y la RCNN, dado que el uso de vectores *one-hot* para codificar los *inputs* conlleva un elevado uso de recursos ya que la dimensión de dichos vectores en los arreglos tridimensionales es de 2^{bits}. El aumento de bits también vuelve la predicción del siguiente número de una cadena un problema más complejo en general, por lo que en esta situación el conjunto de entrenamiento debe ser necesariamente más grande. Hay que tomar en cuenta también que el total de parámetros a entrenar crecería de manera exponencial respecto del número de bits, ya que solo en la capa de *outputs* se requieren 2^{bits} neuronas. Cabe mencionar que no es válido comparar directamente este problema con el ataque sobre el PRNG donde se utilizaron números de 8 bits, ya que predecir el

LCG es una tarea más simple debido a la fuerte interdependencia que existe entre los números pseudoaleatorios dada por la expresión (4.1). Con lo anterior en mente, se identifica que la mayor limitante del presente trabajo es el poder computacional disponible.

Otro aspecto a considerar son las 3 "mejores" instancias de cada tipo de red. Como se mencionó en la sección 6.1, estas fueron las redes 3, 4 y 5 correspondientes a la FNN, LSTM y RCNN, respectivamente. En la selección no tuvo peso relevante el criterio de precisión sobre el conjunto de prueba debido a que, como puede verse en los Cuadros 6.1 a 6.3, no hubo cambios significativos en el desempeño de las redes sobre la data de prueba al variar los parámetros dentro de los límites reportados. Adicionalmente, las diferencias en la precisión entre instancias pueden atribuirse al hecho que existe aleatoriedad al inicializar los pesos y sesgos de las redes al comenzar el entrenamiento. En efecto, al entrenarlas nuevamente desde cero con el mismo conjunto de entrenamiento es posible obtener resultados distintos. No obstante, al tomar en cuenta el criterio de simplicidad, las redes elegidas marcan un punto medio entre un modelo muy simple y muy complicado y esto último es lo que reafirma su validez como las mejores instancias. Una nota relevante respecto la complejidad de los modelos es el largo de *input* L. Para las redes seleccionadas, este no fue mayor a 32, y los resultados sugieren que incrementarlo no otorga mejoras considerables. Sin embargo, no fue posible llegar a utilizar L mayor a 50 dado que implicaría de nuevo un aumento significativo de recursos computacionales. Sería fructífero observar que sucedería empleando L del orden de 100 o mayor, para observar si existen interdependencias a larga distancia dentro del generador.

La elaboración precedente respecto a la aleatoriedad de inicialización de las redes indica que no sería justo basarse solamente en la precisión sobre el conjunto de prueba para determinar cual de los 3 modelos seleccionados tuvo el mejor desempeño sobre la data cruda debido a que todos tuvieron un valor de precisión muy similar. Pueden considerarse entonces las Figuras 6.1 a 6.3. Estas revelan que la RCNN fue la requirió menos *epochs* para que la función de costo convergiera. Sin embargo, su evolución fue errática (i.e. con múltiples picos en la gráfica de costo contra *epoch*). Por otro lado, este modelo también sufrió frecuentemente de *overfit* durante las distintas sesiones de entrenamiento, lo cual señala que es intrínsecamente muy complejo para el problema que se quiere abordar. Por su parte, la LSTM no sufrió tanto de *overfit*, pero similar a la RCNN tuvo una evolución errática. En lo que concierne a la FNN, a pesar de ser la que requirió más *epochs* de entrenamiento tuvo una fase de aprendizaje relativamente más suave que las otras redes. A partir de esto, en el presente estudio, la red que llevó a cabo el mejor análisis de la data cruda fue la red neuronal prealimentada.

Con respecto al análisis de la data posprocesada, ningún modelo de aprendizaje fue capaz de superar consistentemente la probabilidad de adivinar al ser evaluado sobre la data de prueba. Esta afirmación se ve reforzada al tomar en cuenta los resultados de las pruebas estadísticas realizadas que proveyeron valores p que estuvieron por encima del umbral de rechazo en todo caso. Variar la complejidad de los modelos con los números de neuronas, o la longitud del *input* no tuvo efectos notables. Por lo general, durante el entrenamiento, la precisión sobre la data de validación osciló alrededor de $P_A = 0.50$, como se observa en las Figuras 6.4 a 6.6. La evolución de las redes fue errática a excepción del caso de la FNN, donde se tuvo una variación más suave en la gráfica de costo contra *epoch*. Sin embargo, hay que resaltar que esta red particular requirió tanto de *Dropout* como regularización L2 para evitar un *overfit* respecto de la data de entrenamiento. Solamente la LSTM mostró no necesitar regularización durante el entrenamiento. A pesar de esto, ultimadamente el uso de técnicas de mitigación de *overfit* tampoco proveyó mejoras en el desempeño de los modelos de aprendizaje.

Todo esto indica que las secuencias posprocesadas del QRNG cuentan con resistencia contra los modelos de aprendizaje empleados en este estudio. No obstante, es imposible concluir a partir de estos resultados que el generador es completamente inmune a ataques basados en Machine Learning. Múltiples avances se siguen logrando en el área de redes recurrentes, como la LSTM basada en *temporal pattern attention* propuesta por Li et al. (2020), o inicializaciones especiales que llevan a mejores desempeños (Park, 2023). Basándose en esto, para brindar una conclusión firme con respecto a la calidad del generador, es necesario someterlo a más pruebas que involucren ataques realizados por

modelos que sean del estado del arte. Cabe resaltar que en el estudio de la data posprocesada también existe la limitación dada por el poder computacional disponible. Es necesario entonces observar si con acceso a mayor poder computacional es posible lograr resultados distintos. Por ejemplo, podría aumentarse el valor de L a fin de buscar interdependencias a larga distancia, o bien, podría hacerse uso de un conjunto de entrenamiento más grande, cuyo tamaño esté en un orden mayor al de 10^6.

Por último, se tienen los resultados asociados al generador de congruencia lineal. Estos demuestran que las redes poseen capacidades de detectar interdependencias marcadas en secuencias que a primera vista son completamente aleatorias. En efecto, la precisión sobre el conjunto de prueba fue siempre mayor a 0.80 (u 80 %), claramente una mejora respecto a $P_A = 0.0039$. En este caso, la FNN fue la que tuvo el mejor desempeño, llegando a predecir correctamente más del 97 % de los elementos de prueba. Es de notar que este resultado se debe en principio a que el problema del PRNG es más simple que el del QRNG y por ende un modelo más simple de aprendizaje puede proveer un mejor desempeño que modelos que son más complejos.

CAPÍTULO 8

Conclusiones

Haciendo referencia al objetivo general planteado en el capítulo 2 se concluye que los modelos de Machine Learning demostraron un poder predictivo superior al método de adivinar al ser expuestos a la data cruda del generador cuántico. Esto reveló la existencia de correlaciones en el muestreo del convertidor análogo a digital que, en principio, se deben a ruido clásico dentro de la implementación práctica del QRNG. Por otro lado, el análisis de la data posprocesada hizo claro que el generador es resistente a las técnicas de aprendizaje supervisado particulares que se aplicaron en el estudio.

Con respecto a los objetivos específicos, en el mismo orden en el que fueron presentados en el capítulo 2 se concluye lo siguiente:

- Se realizó el estudio del sistema físico asociado al diseño experimental del cual se derivan cotas mínimas para la entropía de los resultados de las mediciones que avalan el uso de esta configuración para extracción de números aleatorios basándose en principios cuánticos.
- Se implementaron las tres arquitecturas de redes neuronales, siendo ellas la FNN, LSTM y RCNN.
- Los resultados sobre el generador de congruencia lineal reafirmaron la capacidad de las redes neuronales de hallar interdependencias en secuencias de números aparentemente aleatorios, lo cual es muestra de la validez de utilizar estos modelos de aprendizaje para llevar a cabo esta clase de estudios criptoanalíticos.

CAPÍTULO 9

Recomendaciones

Como se mencionó, hubo una limitación fuerte debida a los recursos computacionales disponibles, por lo que pueden considerarse múltiples áreas de mejora. En principio, sería fructífero observar el resultado de ataques que puedan utilizar cadenas aleatorias cuyo tamaño esté en un orden mayor al de 10^6 para entrenar y evaluar las redes. Similarmente, hacer uso de longitudes de *input* grandes podría revelar interdependencias a larga distancia en la data. Adicionalmente, existe la posibilidad de que el desempeño de la red mejore a consecuencia de un *binning* con más bits para la data cruda. Todo esto ultimadamente permitiría observar el caso hipotético de un ataque realizado por un adversario con recursos ilimitados, lo cual es deseable en el contexto de criptografía.

Finalmente, se recomienda que a futuro se desarrollen más estudios sobre el generador aplicando técnicas novedosas de Machine Learning como la LSTM basada en *temporal pattern attention* o métodos de inicialización que mejoran considerablemente el aprendizaje de una red neuronal. Es necesario llevar a cabo esta clase de análisis de manera consistente, esto debido a que Machine Learning es una disciplina en constante desarrollo y es de esperarse que cada cierto tiempo se logren descubrimientos nuevos que lleven a modelos más robustos con mayores capacidades de aprendizaje.

CAPÍTULO 10

Referencias

Anderson, D. R., Sweeney, D. J., Williams, T. A., Camm, J. D., y Cochran, J. J. (2014). *Statistics for business & economics*. Cengage Learning.

Blanter, Y. M., y Büttiker, M. (2000). Shot noise in mesoscopic conductors. *Physics reports*, *336*(1-2), 1–166.

Blundell, S. J., y Blundell, K. M. (2010). *Concepts in thermal physics*. Oxford University Press on Demand.

Brown, R. G., Eddelbuettel, D., y Bauer, D. (2018). Dieharder. *Duke University Physics Department Durham, NC (2018)*, 27708–0305.

Chai, Z., Shao, W., Zhang, W., Brown, J., Degraeve, R., Salim, F. D., ... others (2019). Gese-based ovonic threshold switching volatile true random number generator. *IEEE Electron Device Letters*, *41*(2), 228–231.

Chaves, R., Kuzmanov, G., Sousa, L., y Vassiliadis, S. (2006). Improving sha-2 hardware implementations. En *Cryptographic hardware and embedded systems-ches 2006: 8th international workshop, yokohama, japan, october 10-13, 2006. proceedings 8* (pp. 298–310).

ComScire. (s.f.). *Generator selection guide page*. Disponible en: `https://comscire.com/random-number-generator-selection-guide/`

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., y Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. En *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2625–2634).

Fan, F., y Wang, G. (2018). Learning from pseudo-randomness with an artificial neural network–does god play pseudo-dice? *IEEE Access*, *6*, 22987–22992.

Feng, Y., y Hao, L. (2020). Testing randomness using artificial neural network. *IEEE Access*, *8*, 163685–163693.

Ferreira, M. J. (2021). *Quantum-noise based true random number generation* (Tesis de Master). Disponible en: `http://hdl.handle.net/10773/33737`

Ferreira, M. J., Silva, N. A., Pinto, A. N., y Muga, N. J. (2021). Characterization of a quantum random number generator based on vacuum fluctuations. *Applied Sciences*, *11*(16), 7413.

Gabriel, C., Wittmann, C., Sych, D., Dong, R., Mauerer, W., Andersen, U. L., ... Leuchs, G. (2010). A generator for unique quantum random numbers based on vacuum states. *Nature Photonics*, *4*(10), 711–715.

Gerry, C., y Knight, P. (2005). *Introductory quantum optics*. Cambridge university press.

Goldreich, O. (2004). *Foundations of cryptography, basic techniques*. Cambridge university press.
Harrison, R. L. (2010). Introduction to monte carlo simulation. En *Aip conference proceedings* (Vol. 1204, pp. 17–21).
Haw, J.-Y., Assad, S., Lance, A., Ng, N., Sharma, V., Lam, P. K., y Symul, T. (2015). Maximization of extractable randomness in a quantum random-number generator. *Physical Review Applied*, *3*(5), 054004.
Horowitz, P., y Hill, W. (1989). *The art of electronics* (Vol. 2). Cambridge university press.
Hurley-Smith, D., y Hernandez-Castro, J. (2017). Certifiably biased: An in-depth analysis of a common criteria eal4+ certified trng. *IEEE Transactions on Information Forensics and Security*, *13*(4), 1031–1041.
IDQuantique. (s.f.). *Id quantique product page.* Disponible en: `https://www.idquantique.com/random-number-generation/products/`
Ioffe, S., y Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. En *International conference on machine learning* (pp. 448–456).
Katz, J., y Lindell, Y. (2020). *Introduction to modern cryptography*. CRC press.
Ketkar, N., y Moolayil, J. (2021). *Deep learning with python: learn best practices of deep learning models with pytorch*. Springer.
Kim, J., Ahmed, T., Nili, H., Truong, N. D., Yang, J., Jeong, D. S., ... Kavehei, O. (2017). Nano-intrinsic true random number generation. *arXiv preprint arXiv:1701.06020*.
Krawczyk, H. (1995). New hash functions for message authentication. En *Advances in cryptology — eurocrypt'95: International conference on the theory and application of cryptographic techniques saint-malo, france, may 21–25, 1995 proceedings 14* (pp. 301–310).
Li, C., Zhang, J., Sang, L., Gong, L., Wang, L., Wang, A., y Wang, Y. (2020). Deep learning-based security verification for a random number generator using white chaos. *Entropy*, *22*(10), 1134.
Liang, M., y Hu, X. (2015). Recurrent convolutional neural network for object recognition. En *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3367–3375).
Loudon, R. (2000). *The quantum theory of light*. OUP Oxford.
L'Ecuyer, P., y Simard, R. (2013). *A software library in ansi c for empirical testing of random number generators* (Inf. Téc.). Département d'Informatique et de Recherche Opérationnelle Université de Montréal.
Mannalath, V., Mishra, S., y Pathak, A. (2022). A comprehensive review of quantum random number generators: Concepts, classification and the origin of randomness. *arXiv preprint arXiv:2203.00261*.
Meraouche, I., Dutta, S., Tan, H., y Sakurai, K. (2021). Neural networks-based cryptography: A survey. *IEEE Access*, *9*, 124727–124740.
Müller, A. C., y Guido, S. (2016). *Introduction to machine learning with python: a guide for data scientists*. O'Reilly Media, Inc.
Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). Determination press San Francisco, CA, USA.
Park, I. M. (2023). *On asymptotic learning signals in recurrent networks.* Disponible en: `https://mpml.tecnico.ulisboa.pt/seminars?id=6933`
Rényi, A. (1961). On measures of entropy and information. En *Proceedings of the fourth berkeley symposium on mathematical statistics and probability, volume 1: Contributions to the theory of statistics* (Vol. 4, pp. 547–562).
Reyzin, L. (2011). *Extractors and the leftover hash lemma.* Disponible en: `https://www.cs.bu.edu/~reyzin/teaching/s11cs937/notes-leo-1.pdf`
Rukhin, A., Soto, J., Nechvatal, J., Smid, M., y Barker, E. (2001). *A statistical test suite for random and pseudorandom number generators for cryptographic applications* (Inf. Téc.). Booz-allen and hamilton inc mclean va.
Salehinejad, H., Sankar, S., Barfett, J., Colak, E., y Valaee, S. (2017). Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*.
Shultis, J. K., y Faw, R. E. (2016). *Fundamentals of nuclear science and engineering*. CRC press.
Smith, W. F. (2020). *Experimental physics: Principles and practice for the laboratory*. CRC Press.

Toshiba. (s.f.). *Quantum random number generators.* Disponible en: https://www.toshiba.eu/pages/eu/Cambridge-Research-Laboratory/quantum-random-number-generators

Truong, N. D., Haw, J. Y., Assad, S. M., Lam, P. K., y Kavehei, O. (2018). Machine learning cryptanalysis of a quantum random number generator. *IEEE Transactions on Information Forensics and Security*, *14*(2), 403–414.

Zolfaghari, B., y Koshiba, T. (2022). Ai makes crypto evolve. *Applied System Innovation*, *5*(4), 75.

CAPÍTULO 11

Apéndice A

11.1. Código

El código utilizado para la implementación de las redes neuronales, así como los archivos de datos empleados para entrenarlas y llevar a cabo las pruebas estadísticas, pueden hallarse en el siguiente repositorio:

- `https://github.com/cosmarltx/Redes_QRNG.git`

Con el fin de optimizar el uso de recursos se emplearon arreglos de *numpy* en lugar de listas de Python para el manejo de los datos ya que esto reduce significativamente la cantidad de memoria requerida para almacenar información. Similarmente, se utilizó el argumento `dtype = int8` en la construcción de los arreglos a fin de minimizar su espacio en memoria.

Por otro lado, keras permite una aceleración significativa del proceso de entrenamiento haciendo uso de una *unidad de procesamiento gráfico* o GPU por sus siglas en inglés. Sin embargo, el ordenador empleado en el estudio no contó con una tarjeta GPU compatible con keras. A raíz de esto, se decidió utilizar los aceleradores proveídos gratuitamente en *Google Colab*, a cambio de una memoria RAM más limitada (i.e. menos recursos de memoria). Estos proveyeron sesiones de entrenamiento más dinámicas a cambio de una limitación de la memoria disponible.

CAPÍTULO 12

Apéndice B

12.1. Prueba estadística

Con el fin de validar estadísticamente los resultados, estos se sometieron a una prueba estadística. Para esto, se consideraron dos métodos cuyo propósito fue predecir el siguiente elemento de la secuencia aleatoria. El método A consistió en aplicar la red neuronal, mientras que el método B se basó en adivinar. Se crearon 500 grupos con 4000 números que hay que predecir. Para la prueba, se anotó cuantos de los 4000 números se lograron predecir correctamente con cada método. Así, se construyeron 500 muestras emparejadas, donde cada una estuvo constituida por el total de aciertos de los métodos A y B sobre el mismo grupo de 4000 cifras.

Partiendo de lo anterior, se propuso determinar si la media del *número de aciertos* del método A era superior a la del método B. Por lo tanto, se establecieron las siguientes hipótesis:

$$H_o : \mu_A \leq \mu_B$$
$$H_a : \mu_A > \mu_B$$

Se utilizó la *prueba t de muestras emparejadas* con significancia estadística $\alpha = 0.001$. A continuación se desarrolla como ejemplo el proceso realizado para el caso de la red neuronal prealimentada de la data cruda. Cabe mencionar que este procedimiento fue el mismo para las otras cinco pruebas reportadas en la sección de resultados.

En primer lugar, es necesario demostrar que las diferencias entre el total de aciertos de las muestras emparejadas siguen una distribución normal, ya que esto es una suposición de la prueba (Anderson et al., 2014). En efecto, puede observarse que el histograma de dichas diferencias en la Figura 12.1 cuenta con la forma de campana de la distribución gaussiana. Por otro lado, en la Figura 12.2 se incluye el gráfico Q-Q de las diferencias donde se tiene que los puntos siguen una recta. Tanto esto, como el hecho que la curtosis y el coeficiente de asimetría de la distribución son cercanos a cero, con valores de -0.210 y 0.122 respectivamente, son indicadores fuertes de que las diferencias están distribuidas normalmente.

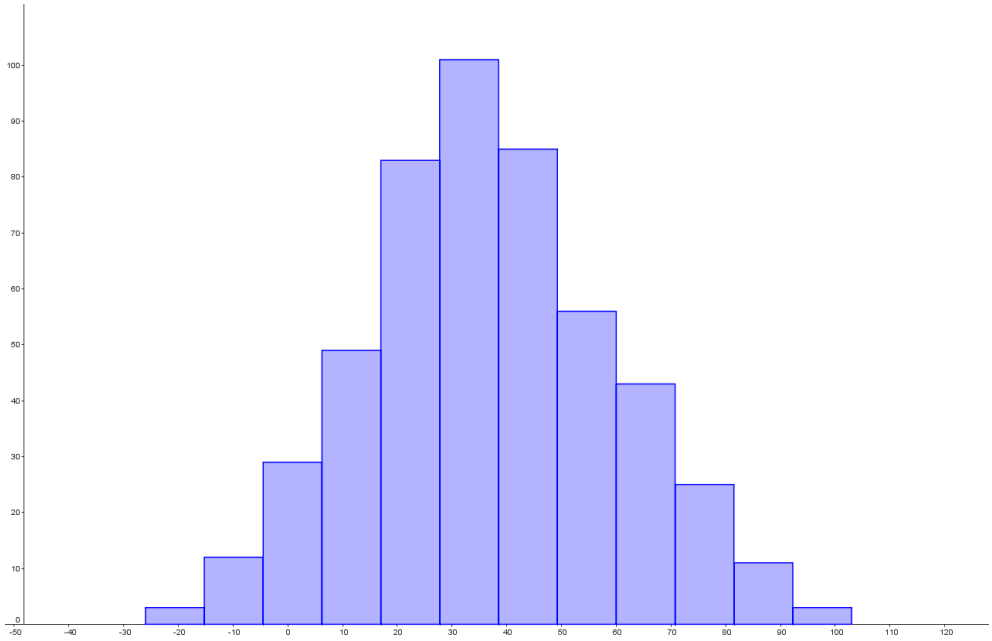

Figura 12.1: Histograma de las diferencias de las muestras emparejadas
Fuente: elaboración propia

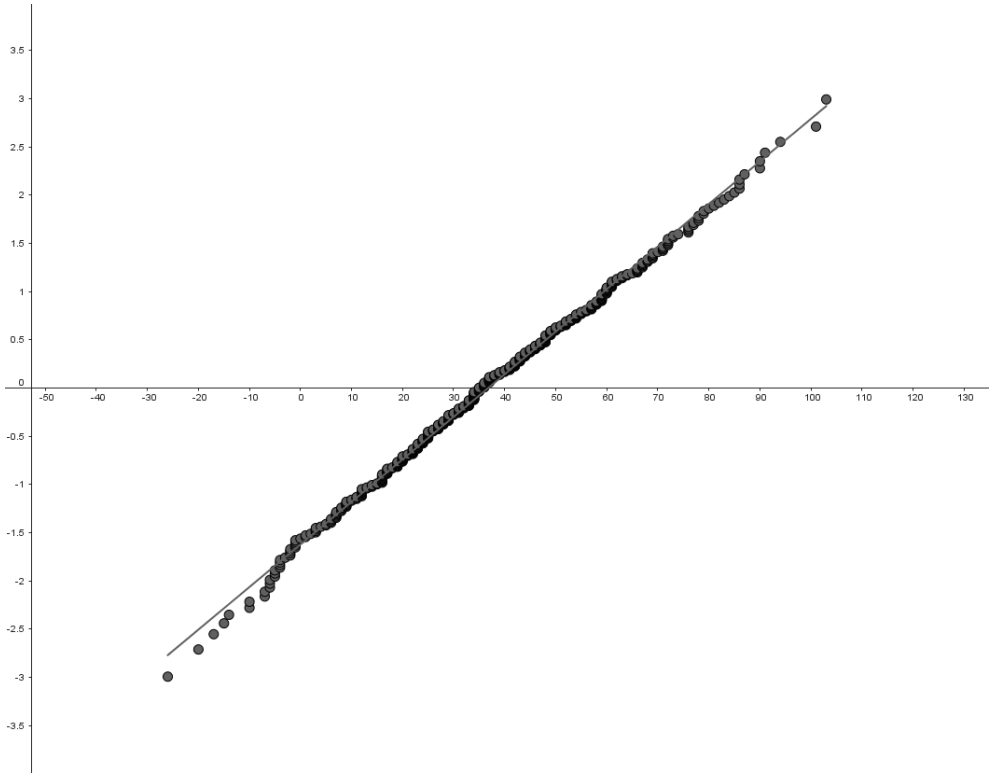

Figura 12.2: Gráfico Q-Q de las diferencias de las muestras emparejadas
Fuente: elaboración propia

Después de demostrar la normalidad de los datos, se procede a emplear el programa Microsoft Excel con la herramienta *Análisis de datos*. Esta incluye una función de prueba t para muestras emparejadas, la cual, entre otras cosas, provee el valor p de una y dos colas. Por la forma de las hipótesis, en este caso se emplea el valor p de una cola, que resultó ser 3.031×10^{-142}, el cual es menor que la significancia 0.001. Por ende, se cuenta con suficiente evidencia estadística para rechazar H_o y se concluye que en promedio, la red prealimentada logra un mayor número de aciertos que el método de adivinar.