UCUENCA

Universidad de Cuenca

Facultad de Ingeniería

Carrera de Ingeniería en Telecomunicaciones

Uso de Deep Learning para la codificación y decodificación en canales de una sola vía

Trabajo de titulación previo a la obtención del título de Ingeniero en Telecomunicaciones

Autores:

Juan Sebastián Álvarez Villavicencio

Edgar Mateo Bacuilima Crespo

Director:

Kenneth Samuel Palacio Baus

ORCID: 0000-0002-7318-8062

Cuenca, Ecuador

2024-08-26

Resumen

Este trabajo evalúa la aplicación de técnicas de aprendizaje profundo en el descubrimiento de códigos para canales unidireccionales, específicamente en un Binary Symmetric Channel (BSC). El objetivo principal es crear códigos comparables con las técnicas de codificación convencionales, enfrentando los retos que plantea la inclusión de aprendizaje automático en el proceso. Este documento revisa exhaustivamente trabajos relacionados y el marco teórico necesario para comprender el problema, cubriendo redes neuronales y códigos convencionales en canales binarios simétricos. Luego, se proponen experimentos con redes neuronales profundas para crear modelos de codificador y decodificador de canal, describiendo las arquitecturas, tasas de codificación y aspectos técnicos clave. Se emplean capas lineales, redes Gated Recurrent Unit (GRU) y Long Short-Term Memory (LSTM). Los resultados experimentales que se derivan de cada experimento permiten observar de manera gráfica el desempeño de los códigos descubiertos por los modelos neuronales frente a códigos de corrección de errores convencionales en términos de la medición del Bit Error Rate (BER) para determinados valores de q (probabilidad de error de bit del canal). Finalmente, se discuten las limitaciones inherentes al canal binario simétrico y su impacto en el desarrollo y los resultados de este trabajo. Además, se esbozaron líneas de investigación futuras que podrían contribuir a mejorar los resultados, incluyendo una interpretación de los códigos descubiertos en el marco de la Teoría de la Codificación.

Palabras clave del autor: aprendizaje profundo, codificación, redes neuronales, códigos convencionales, canal binario

Abstract

This work evaluates the application of deep learning techniques in the discovery of codes for one-way channels, specifically in a BSC. The main objective is to create codes comparable to conventional coding techniques, addressing the challenges posed by the inclusion of machine learning in the process. This document provides an exhaustive review of related work and the theoretical framework necessary to understand the problem, covering neural networks and conventional codes in binary symmetric channels. Then, experiments with deep neural networks are proposed to create encoder and decoder models for the channel, describing the architectures, coding rates, and key technical aspects. Linear layers, GRU, and LSTM networks are employed. The experimental results derived from each experiment graphically show the performance of the codes discovered by the neural models compared to conventional error-correcting codes in terms of BER measurement for specific values of q (the bit error probability of the channel). Finally, the inherent limitations of the binary symmetric channel and their impact on the development and results of this work are discussed. Additionally, future research directions that could help improve the results are outlined, including an interpretation of the discovered codes within the framework of Coding Theory.

Author Keywords: deep learning, coding, neural networks, conventional codes, binary channel

Índice general

Índice de figuras

Índice de tablas

Dedicatoria

Quiero dedicar este trabajo a:

A mis padres, Patricio y Giselle, quienes han sido figuras fundamentales en mi desarrollo personal y modelos a seguir en todos los aspectos posibles. Gracias por ser mis primeros y más importantes maestros, fuentes de inspiración y ejemplos de sacrificio, dedicación y perseverancia. Su tolerancia y apoyo incondicional, así como su ayuda en los momentos más complicados de mi vida, me han brindado la mejor vida que una persona podría tener.

A mi hermana María Emilia, quien ha sido un apoyo constante en mi camino universitario. Siempre me ha brindado consejos sabios, cariño y apoyo de manera incondicional. Su presencia ha sido fundamental en los momentos de oscuridad. Gracias a ella, he aprendido a ser mejor ser humano, cultivando la empatía. Su ejemplo de dedicación y generosidad ha sido una guía invaluable en este recorrido.

A mi compañero de tesis y amigo Mateo. Su apoyo incondicional y su amistad han sido fundamentales para mí, estando a mi lado en los momentos más difíciles, ofreciendo su ayuda y ánimo. Su dedicación y esfuerzo han sido elementos importantes para la culminación de este trabajo y de la carrera. Estoy profundamente agradecido por su colaboración, su compromiso y su amistad.

A mi tutor Kenneth, quien, con su habilidad para dar clases y motivar, me ha mejorado como estudiante y ser humano, además de impulsar y alimentar mis ganas de aprender. Este trabajo va dedicado hacia aquel profesor que siembra inspiración constantemente en quienes son afortunados de ser sus estudiantes

Juan Sebastian Álvarez Villavicencio

Dedicatoria

Este trabajo, todo el proceso y más que nada el esfuerzo y perseverancia que me han llevado hasta este punto va dedicado a mi familia. Para mis padres Marco y Jovvhanca, por haberme brindado todas las facilidades y herramientas, las cuáles he intentado aprovechar al máximo para que su sacrificio valga la pena.

A mi hermano Max, quien ha sido mi ejemplo ha seguir durante toda mi vida y me ha brindado su apoyo incondicional en cada decisión que he tomado. Por sus consejos, regaños y sobre todo por siempre intentar impregnar en mí el deseo de ser alguien mejor para este mundo, cada gramo de empeño que he dirigido a esta etapa de mi vida te lo dedico.

Así mismo, para mis sobrinos Maximiliano y Enzo, que espero algún día vean en mí un ejemplo a seguir, les dedico cada idea que se plasma en este documento y que les sirva de inspiración para que superen los retos que les espera a lo largo de su vida.

También dedico este trabajo a mi amigo y compañero de tesis Sebastián, con el objetivo de recordarle que no importa las dificultades que se le presenten en la vida, pues tal como lo hice en cada trabajo que desarrollamos en esta etapa académica, confió plenamente en sus habilidades y que sabra superarlas con facilidad. No dudes jamás en que siempre estaré dispuesto a estrechar mi mano de apoyo si lo necesitas.

Finalmente, dedico esta investigación a nuestro tutor Kenneth, pues espero que este trabajo sea un reflejo del excelente docente, educador y ser humano que es y que tome este trabajo como inspiración para continuar formando profesionales, sin olvidarse que detrás de cada alumno existe una persona, tal como lo ha hecho con nosotros.

Edgar Mateo Bacuilima Crespo

Agradecimientos

Quiero expresar mi más profundo agradecimiento a nuestro tutor, Kenneth. Su guía y su disposición para compartir conocimientos han sido los pilares fundamentales sobre los que se ha construido este trabajo. Su habilidad para explicar de manera sencilla temas complejos ha sido crucial para que este proyecto se haya podido llevar a cabo con éxito.

Asi mismo, me gustaría agradecer a mi compañero de tesis y amigo, Mateo. Su apoyo y ayuda han sido esenciales para superar todas las dificultades que se han presentado durante la realización de este trabajo. Le agradezco tanto por su apoyo académico como emocional, y por su constante disposición para trabajar arduamente con el objetivo de sacar adelante no solo este proyecto, sino todos los trabajos que hemos realizado juntos.

Finalmente, quiero agradecer a mi familia, por su incansable apoyo y por creer en mí en todo momento. Su cariño, comprensión y consejos me han permitido mantenerme enfocado en la realización de este trabajo de manera constante y continua.

Juan Sebastián Álvarez Villavicencio

Agradecimientos

Agradezco, a nuestro tutor Kenneth por haber sido la guía que ha iluminado el camino para el desarrollo de este trabajo de titulación, sin su orientación y apoyo sin duda el trayecto habría sido más complicado. Así mismo, le doy la gracias por representar un ejemplo a seguir para cada uno de sus estudiantes y sobre todo por demostrar interés genuino en el desarrollo académico de cada uno de nosotros, no cabe duda que sus conocimientos y disposición por compartirlos han marcado un antes y después en el transcurso de esta etapa universitaria.

A mi amigo y compañero de tesis Sebastián, por haber representado el apoyo no solo en el desarrollo de este trabajo, sino en cada una de las actividades que hemos llevado a cabo durante todo este tiempo, por siempre ser una persona confiable, comprensiva y sobre todo por haberme brindado su amistad y lealtad simplemente estoy completamente agradecido. No me imagino haber transcurrido este viaje con nadie más, muchas gracias.

Finalmente, agradezco a mi familia, por haberme brindado el apoyo emocional y jamás dejar que me diera por vencido. A mis padres por mantenerme fuerte en cada paso que he dado a costa de su sacrificio y a mi hermano Max por la confianza total en momentos donde en mí escaseaba, realmente eso ha marcado completamente la diferencia y han hecho que llegue hasta aquí, gracias totales.

Edgar Mateo Bacuilima Crespo

Abreviaciones y acrónimos

AA Aprendizaje Automático. 46

ADAM Adaptive Moment Estimation. 67, 79

ANN Artificial Neural Network. 46, 57, 59

ARQ Automatic Repeat reQuest. 29

AWGN Additive White Gaussian Noise. 21–24, 33, 36, 45, 61, 62, 80, 82, 86, 89, 139

B-DMC Binary Discrete Memoryless Channel. 44

BCE Binary Cross Entropy. 67

BCH Bose–Chaudhuri–Hocquenghem (Códigos). 59, 70

BCJR Bahl, Cocke, Jelinek, and Raviv (Algoritmo). 37, 38

BEC Binary Erasure Channel. 22

BER Bit Error Rate. 2, 3, 7, 21, 22, 26, 32, 36, 38, 43, 46, 60, 61, 63, 64, 69, 70, 80, 83, 85–87, 91, 97, 104, 105, 107, 110, 111, 115, 117, 121, 122, 129, 131, 133, 136

BPSK Binary Phase Shift Keying. 21, 92, 134, 135

BSC Binary Symmetric Channel. 2, 3, 7, 22, 24, 25, 30–32, 43, 45, 62, 65, 82, 86, 89, 133, 135, 136, 139, 140

CNN Convolutional Neural Network. 58, 69–71

CPU Central Processing Unit. 73, 75, 112, 135

CUDA Compute Unified Device Architecture. 112, 135

DEF Deep Extended Feedback. 7, 66–68

DNN Deep Neural Network. 57, 66, 67, 70, 136, 137

DRN Doubly Residual Neural. 69, 70

FAIR Facebook AI Research. 141

FCN Fully Connected Network. 69

FEC Forward Error Correction. 19, 29

GPU Graphics Processing Unit. 24, 26, 73, 75–77, 112, 135

GRU Gated Recurrent Unit. 2, 3, 56, 67, 99, 109, 122, 136

HARQ Hybrid automatic repeat request. 66

IA Inteligencia Artificial. 8, 18, 23, 26, 46, 71, 72

IoT Internet of Things. 18

IQ “In-Phase” y “Quadrature”. 58

LDPC Low-Density Parity-Check. 19, 21, 33, 62, 69, 70, 79, 80

LLR Log-Likelihood Ratio. 45

LSTM Long Short-Term Memory. 2, 3, 7, 9, 55, 56, 67, 68, 100, 109, 122, 124, 132, 133, 136, 137

LTE Long Term Evolution. 61, 69

MSE Mean Squared Error. 80

NN Neural Network. 80

ONNX Open Neural Network Exchange. 77, 78

PAM Pulse Amplitude Modulation. 67

PSG Parity Symbol Generator. 67

QAM Quadrature Amplitude Modulation. 67

RAM Random Access Memory. 112

RCBQ Random Codes Based on Quasigroups. 42

ReLU Rectified Linear Unit. 51

RNG Random Number Generator. 43

RNN Recurrent Neural Network. 53–56, 62, 63, 67, 71, 100

SC Successive Cancellation. 45

SEM Standard Error of the Mean. 111

SGD Stochastic Gradient Descent. 70

SNR Signal-to-Noise Rati. 7, 21, 22, 37–39, 63, 64, 86

XOR Exclusive OR. 41, 51, 88–90

Capítulo 1 Introducción

En este capítulo se describe de manera general, el enfoque de este trabajo de titulación, principalmente motivado por el reciente impulso que ha tomado la IA en diferentes aplicaciones, y particularmente en el campo de la codificación de canal. La Sección 1.1, aborda la identificación del problema, en la que se describe el contexto en el cual se centra el desarrollo del proyecto y se enfatiza la necesidad de explorar nuevas técnicas de codificación. Seguidamente, la Sección 1.2 describe el enfoque propuesto para abordar el problema de la codificación de canal. En la Sección 1.3, se indica el alcance del trabajo de titulación en términos de lo que se espera obtener como resultado de la implementación de nuestra propuesta. Finalmente, la Sección 1.4 presenta tanto el objetivo general como los objetivos específicos propuestos. Este capítulo brinda una visión general del trabajo que se ha desarrollado y se expone la relevancia que tiene en el contexto de los sistemas de comunicación y de la teoría de la codificación de manera particular.

1.1. Identificación del problema

El constante crecimiento del tráfico de información en las redes de comunicación actuales debido principalmente a la masificación del uso y dependencia de los servicios de telecomunicaciones en la población, ha motivado a diferentes organizaciones, empresas y especialmente a la academia, a buscar nuevas formas eficientes y confiables de transmisión de información. Este incremento del tráfico ha sido acelerado en años recientes debido al nacimiento de nuevas tecnologías de telefonía celular como 5G, o del Internet of Things (IoT), etc. En las que la cantidad de información transmitida exige sistemas más robustos y capaces de transmitir grandes volúmenes de datos en tiempo real. Por otro lado, el principal desafío en la transmisión de información a través de distintos canales se centra en la detección y corrección de errores, que se ha constituido como un área de intensa investigación durante varias décadas. En 1948, Shannon [12] demostró que al codificar la información de manera adecuada, es posible reducir la probabilidad de errores en la transmisión producidos por interferencias o ruido en el canal de comunicación, incluso hasta llegar asintóticamente a cero [13]. Desde la publicación de este artículo de Shannon, se han llevado a cabo esfuer-

zos mayúsculos para la generación de sistemas de codificación y decodificación cada vez más eficientes. Actualmente existe un conjunto de códigos ya implementados en distintas tecnologías y generaciones de comunicación tanto móviles como fijas estos incluyendo Turbo Códigos [14] (3G y 4G), Low-Density Parity-Check (LDPC) [15] 4G, códigos convolucionales (2G y 3G) [16], códigos de Hamming (2G) [17], códigos polares (5G NR) [18], etc.

Todos estos códigos se engloban dentro de la categoría de códigos Forward Error Correction (FEC). Al usar códigos FEC, el elemento encargado de transformar el mensaje original a ser transmitido en una palabra codificada y capaz de incrementar la robustez de la integridad de la información enviada recibe el nombre de codificador de canal. Así el equipo transmisor toma un mensaje usualmente representado como un número binario de m bits y agrega redundancia (mediante la inclusión de otros bits) con el fin de producir una palabra codificada. Esta palabra puede enviarse por el canal para que al ser recibida en el destino pueda decodificarse correctamente, identificando el mensaje enviado originalmente. Convencionalmente, el rendimiento de un sistema de comunicaciones se mide en términos de una métrica como la tasa de error por bit transmitido, la cual se espera reducir explotando la redundancia introducida por el codificador. El efecto combinado de incorporar un decodificador y codificador es el de reducir las indeseables consecuencias del ruido en el canal sobre la información transmitida, que se entiende como interferencias de diversas índoles. El uso de FEC se fundamenta en el Teorema de Codificación de Canal propuesto por Shannon, y puede resumirse en un uso controlado de redundancia en la palabra de código transmitida, tanto para detectar errores como para corregir los errores causados durante la transmisión por un canal ruidoso. Sin importar si el resultado de la decodificación es o no correcto, no se realiza ningún procesamiento adicional en el receptor [19] a este nivel de comunicación de capa física.

Actualmente, se han desarrollado múltiples códigos para diferentes modelos de canales, sin embargo, algunos experimentan en la práctica, el problema de requerir extensas tablas de búsqueda, lo que dificulta su implementación debido a la pesada carga computacional que implica la exploración de dichas tablas. Desde la perspectiva de ingeniería el problema se reduce a la búsqueda de códigos que permitan codificación

y decodificación en una cantidad razonable de tiempo (complejidad computacional) y que al mismo tiempo, permitan obtener tasas de error promedio por bit por debajo de ciertos límites de tolerancia. En particular, el factor de tiempo de procesamiento es el principal obstáculo en la meta de alcanzar los límites impuestos por la teoría de Shannon, ya que un código debería tener palabras de longitud infinita. No obstante, hoy en día existen códigos muy eficientes para diversos tipos de canales ruidosos en general. Existen códigos que pueden lograr una probabilidad de error muy baja, siempre que la cantidad de información transmitida (denotada como tasa de transmisión o *rate*) esté por debajo de la máxima tasa de transmisión establecida por la capacidad del canal. Así mismo existen códigos que solamente pueden operar satisfactoriamente con tasas de transmisión de cero. Por último, existen los códigos prácticos que son códigos que pueden codificar y decodificar eficientemente en tiempo y espacio polinómicos en función de la longitud del bloque (que hace referencia al tamaño de las palabras generadas por el elemento codificador) [20].

Para el caso de canales cuyo ruido es caracterizado por una variable aleatoria con una distribución Gaussiana y con la característica de ser aditivo a la señal transmitida (Additive White Gaussian Noise), se han desarrollado múltiples códigos que actualmente se utilizan en la práctica en los modernos sistemas de comunicaciones. En [20] encontramos un par de códigos muy populares, empezando por los códigos convolucionales estándar, los cuales se usan ampliamente en comunicaciones satélites, adicionalmente se menciona los códigos convolucionales concatenados, que se basan en el uso de los códigos convolucionales estándar como código interno en una estructura concatenada donde el código externo es un código Reed-Solomon [20] con símbolos de ocho bits, presentando un enfoque más robusto y ampliamente utilizado en entornos de comunicación como el espacio profundo.

Otra familia de códigos muy utilizados corresponde a los Códigos Turbo o *Turbocodes*, los cuales fueron introducidos en 1993 en [20]. Se fundamentan en el uso de dos codificadores basados en códigos convolucionales alimentados por los bits que componen el mensaje original a ser transmitido. Estos codificadores transmiten los bits de paridad de cada código constituyente. La decodificación de estos códigos implica un proceso iterativo de decodificación de cada uno de los códigos constituyentes.

En este punto, de manera análoga al transmisor, se usan dos decodificadores, con la diferencia de que, los decodificadores funcionan en serie.

De manera general, cada código desarrollado presenta ventajas y desventajas, principalmente asociadas al tipo o modelo de canal sobre el cual se aplican, puesto que éste impone condiciones físicas y restricciones inquebrantables al codificador. Como se ha mencionado, existe una gran cantidad de códigos que pueden ser usados con el objetivo de transmitir información con una baja tasa de errores, sin embargo, la respuesta definitiva sobre la optimalidad de los códigos usados para diferentes canales continúa como un trabajo en curso. Hoy en día, el rápido crecimiento de las redes de comunicaciones y del número de usuarios y dispositivos que conforman nuestro modo de vida presente implica transmitir bastas cantidades de información de manera confiable. Dicha confiabilidad se relaciona con una transmisión robusta ante la ocurrencia de errores, es decir, que posibilite una recuperación de la información recibida de manera incorrecta, y en lo posible contribuya a disminuir la congestión de las redes con mensajes de repetición. En ese contexto se necesita una constante búsqueda de nuevos códigos o del mejoramiento de los códigos actuales de corrección de errores con el objetivo de que estos vayan a la par del ritmo de crecimiento de la demanda de tráfico de información actual.

Cada código tiene diferentes características y rendimiento. Un análisis interesante se presenta en [21], donde se realiza una comparativa entre códigos convolucionales, Turbo códigos, LDPC y códigos Polares, los cuales se han utilizado ampliamente en las tecnologías celulares más recientes. Para esta comparación se ha utilizado un esquema de modulación de desplazamiento de fase binaria o Binary Phase Shift Keying (BPSK) sobre un canal Additive White Gaussian Noise (AWGN). Un hecho importante es que el uso de códigos mejora sustancialmente las tasas de error por bit transmitido en comparación con una transmisión de datos sin codificación. En el estudio realizado en [21], se observa que los códigos simulados se comportan de manera similar para diferentes SNR, con la única excepción de los códigos convolucionales. A pesar de que estos últimos tienen un BER menor que los datos sin codificación, su desempeño es peor en comparación con el resto de los códigos. Además, todos los códigos analizados, exceptuando los códigos convolucionales, muestran una respuesta muy similar

cuando se incrementa la tasa de información y el tamaño del bloque de codificación (rate).

En [22] se presenta una comparación centrada en códigos Polares y Turbo códigos donde se destaca la generación de códigos polares como un método nuevo de codificación para la fecha de la publicación. En esta comparación se constata lo mostrado en [21], puesto que estos dos códigos se comportan de manera muy similar ante un determinado nivel de SNR, sin embargo, podemos notar que a medida que se aumenta el tamaño de bloque, los códigos polares presentan un mejor rendimiento que los Turbo códigos.

La construcción de cada código está usualmente relacionada al modelo de canal sobre el cual va a utilizarse, dado que en muchas ocasiones, su operación se ve restringida por las características físicas del canal. No obstante, una práctica común es evaluar su operación en diferentes modelos. Así, tal como el caso de canales AWGN, estos códigos también han sido evaluados en canales discretos cómo BSC y Binary Erasure Channel (BEC). Según se menciona en [22], los turbo códigos experimentan dificultades cuando se usan en canales BSC y BEC debido a que son canales de "salida fuerte" lo que hace que la comunicación entre decodificadores no sea confiable y por lo tanto existan errores transmitidos entre decodificadores. A pesar de esto, podemos observar como para niveles indicados de probabilidad de error, los Turbo códigos funcionan con un nivel de BER bastante bueno en estos canales. El uso de estos códigos en canales binarios se revisa con mayor profundidad en [23] y [24].

Si bien estos códigos presentan un rendimiento aceptable, la búsqueda de nuevos códigos sigue abierta, y con el fin de obtener mejoras tanto en la carga computacional de su operación como en la obtención de mejores tasas de error.

1.2. Propuesta y Justificación

La constante problemática que representa la necesidad de tasas de transmisión cada vez más altas y la limitación del ingenio humano para desarrollar códigos prácticos de corrección de errores capaces de adaptarse y asegurar la transmisión de datos, induce la investigación y creación de nuevos métodos de codificación capaces de sa-

tisfacer los requerimientos de las nuevas generaciones. En este proyecto, se propone la aplicación de técnicas de Aprendizaje Profundo (Deep Learning) para abordar esta problemática y generar códigos cuyos resultados puedan compararse con algunos de los códigos convencionales mencionados en la sección anterior. Esta estrategia ha sido exitosamente probada en canales AWGN con retroalimentación, lo que se puede apreciar en un conjunto de investigaciones relacionadas a este tema tal como [7].

Actualmente, existe un conjunto de modelos y arquitecturas de aprendizaje que se pueden explorar en diferentes clases de canales de comunicación, sin embargo, esta propuesta se enfoca en canales binarios. La elección de este tipo de canales se fundamenta en su enfoque didáctico e inherente practicidad, como la comunicación en sistemas de almacenamiento, tales como unidades de disco duro o memorias digitales y ciertos sistemas de telecomunicaciones en general que pueden ser modelados mediante estos canales.

Así mismo, dirigimos nuestro enfoque a canales binarios puesto que hemos notado que en mayor parte este tipo de técnicas han sido empleadas en canales AWGN, los cuales, también son muy importantes y con múltiples aplicaciones prácticas. Sin embargo, estos canales no abarcan ni representan el universo de medios de comunicación existentes, por lo que se desea alumbrar zonas aún no expuestas centrándonos en canales que tienen un funcionamiento tanto físico como matemático diferente a los canales AWGN. Como se mencionó anteriormente, los canales binarios requieren que las señales presentes en el canal adopten únicamente dos valores, en contraste con un AWGN en el que las señales a ser transmitidas corresponden al conjunto de números reales.

Finalmente, este trabajo también nace con la intención de explorar el área de investigación relacionada con el descubrimiento de códigos para la transmisión de información utilizando IA. El uso de técnicas de Deep Learning en la búsqueda de estos códigos abre nuevas oportunidades y sugiere un nuevo horizonte en el que la IA sirva de soporte para el desarrollo de códigos. Se espera que el enfoque de este trabajo de titulación en esta área, contribuya en el mejoramiento de las características de transmisión de información en futuras generaciones de comunicación.

1.3. Alcance

El desarrollo experimental de este trabajo involucra un camino de investigación, estudio y aplicación de diferentes arquitecturas y técnicas de aprendizaje profundo para la creación de codificadores y decodificadores. Así mismo, involucra el uso de métodos prácticos y de optimización para su implementación sobre canales binarios.

En general, el proyecto se puede seccionar en cuatro etapas, en primer lugar estudiar el modelo de canal de comunicación. En particular, se utiliza un canal simétrico binario BSC, con la meta de explorar un modelo de canal con y sin retroalimentación ruidosa (y perfecta), es decir, que en la arquitectura experimental diseñada, el transmisor sea capaz de recibir la información que obtiene el receptor luego de atravesar el canal, esto debido a que, la información de retroalimentación puede resultar extremadamente útil para sistemas de aprendizaje basados en redes neuronales dentro de la fase de entrenamiento, como ya se ha comprobado para un AWGN en [7].

En la segunda etapa, se llevará a cabo la implementación de un codificador y un decodificador mediante sistemas basados en redes neuronales y aprendizaje supervisado, donde se produce un entrenamiento simultáneo del codificador y el decodificador. Este desarrollo será implementado en lenguaje Python, debido a la sencillez del idioma con respecto a la sintaxis y estructuras para el desarrollo de programación científica, además cuenta con múltiples librerías disponibles para la creación y manejo de redes neuronales denominada Pytorch, la cuál simplifica en gran medida la adaptación de la información para que sea procesada por un procesador potente Graphics Processing Unit (GPU). Este último detalle es muy importante ya que en algunos estudios se muestra que la ejecución de la fase de entrenamiento empleando una GPU produce mejores resultados puesto que estos dispositivos están diseñados para operar con matrices de gran tamaño y que poseen gran cantidad de información, tal como se realiza en este proyecto. Si la arquitectura experimental tiene retroalimentación se presenta un escenario como el de la Figura 1.1.

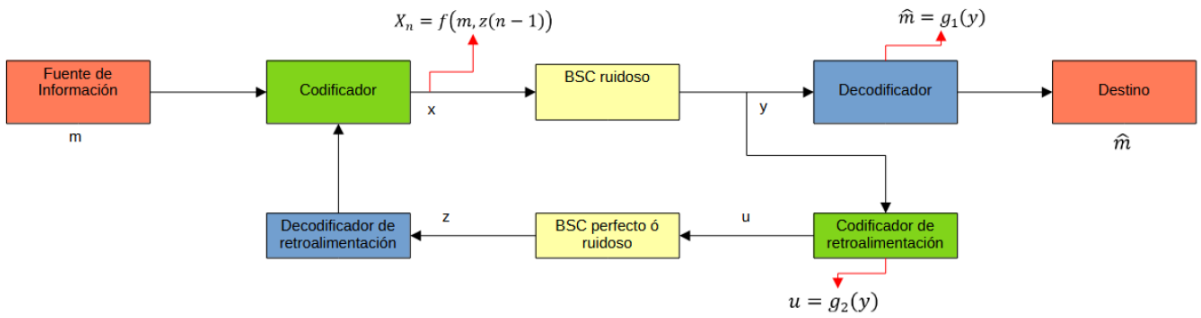

Figura 1.1: Esquema del modelo de entrenamiento y simulación planteado.

Para el proceso de entrenamiento, se creará un conjunto de datos representando una fuente de información que contendrá una cantidad específica de mensajes para la transmisión, dicha fuente será generada de manera aleatoria, para simular una comunicación convencional entre un transmisor y receptor a través de un BSC. De este conjunto, una parte se utilizará para el entrenamiento simultáneo de los codificadores y decodificadores, mientras que el resto se reservará para las pruebas de funcionamiento, respectivamente en los conjuntos de validación y pruebas.

De acuerdo con la Figura 1.1 (en la que el canal de retroalimentación puede omitirse para el caso sin feedback) cada mensaje ingresa al codificador, cuya arquitectura está basada en una red neuronal, dicha arquitectura será seleccionada luego de una etapa de prueba y error hasta alcanzar el mejor modelo tanto para el codificador como para el decodificador. Los modelos de redes neuronales en los que se basan los diseños del codificador y decodificador inician con distintos parámetros (o pesos que caracterizan las conexiones entre los elementos neuronales de procesamiento) con valores asignados de manera aleatoria. El codificador recibe como entrada el mensaje a transmitir y en caso de disponer de retroalimentación, cuenta adicionalmente con otras entradas destinadas a las señales recibidas a través del canal de retroalimentación. La salida del codificador corresponde a una nueva palabra codificada en función del mensaje a enviar. Una vez que el codificador ha generado la palabra a transmitir, ésta se introduce al BSC para ser recibida por el decodificador, el cual produce una salida que, en caso de una decodificación correcta corresponde al mensaje transmitido originalmente.

Este proyecto ha optado por el uso de aprendizaje supervisado, el mismo que per-

mite que los mensajes decodificados puedan compararse con los originales, de tal modo que el resultado de esta comparación se utilice para la corrección progresiva de los pesos del modelo neuronal tanto del decodificador como del codificador para la próxima iteración. De forma simultánea, y ante la presencia de un enlace de retroalimentación, se sigue un proceso similar, y la información decodificada se convierte en el parámetro de entrada al codificador para ajustar los pesos nuevamente. Este ciclo se repite de manera iterativa hasta que se agoten los mensajes del conjunto de entrenamiento. Se ha previsto que los procesos experimentales se ejecuten sobre la plataforma Google Colab [1], debido al requerimiento de una buena capacidad de procesamiento para el entrenamiento del codificador y decodificador, así como de las distintas simulaciones que se deben llevar a cabo.

Tras la culminación del entrenamiento, el modelo obtenido se somete a un nuevo conjunto de datos de prueba, con el objetivo de realizar una comparación entre los resultados del BER (utilizado como una métrica práctica de comparación) que se consigue empleando el método propuesto basado en aprendizaje profundo, frente a códigos convencionales como los Turbo códigos, códigos convolucionales, entre otros. Con este fin, en esta etapa se ejecuta la evaluación de estos códigos convencionales en condiciones similares de tasa de bits y de modelo de canal.

1.4. Objetivos

1.4.1. Objetivo general

Evaluar la aplicación de técnicas de Deep Learning para el descubrimiento de códigos en canales de una sola vía.

[1] En [25] indican que Google Colab es una herramienta diseñada por Google, que tiene el objetivo de otorgar acceso a GPU y TPU a cualquier persona, en cualquier lugar del mundo. Es una herramienta cuyas funciones se centran específicamente en el desarrollo de aplicaciones de análisis de datos e IA. Puede considerarse como una versión avanzada de Jupyter Notebook en cuanto a su interfaz y funcionalidad.

1.4.2. Objetivos específicos

El presente trabajo de titulación posee los siguientes objetivos específicos.

- Estudiar diferentes arquitecturas y técnicas de aprendizaje profundo (Deep Learning – DL) para la implementación de codificadores y decodificadores en canales de una sola vía.
- Implementar un sistema de aprendizaje (DL) para el descubrimiento de códigos.
- Comparar el rendimiento de los códigos descubiertos mediante DL con respecto a códigos convencionales.

Capítulo 2 Marco Teórico

2.1. Introducción

En este capítulo, se presentan principios fundamentales que han sido revisados para el desarrollo e implementación del trabajo planteado. Esta revisión se centra en tres temas fundamentales que son: el teorema general de la codificación de canal ruidoso, técnicas de codificación convencionales y técnicas de codificación basada en redes neuronales.

Con este fin se presenta una revisión del estado del arte, en el que se estudia diferentes artículos y trabajos relacionados que contribuyen al desarrollo y planteamiento de la metodología de propuesta en este trabajo. De igual forma, los resultados experimentales de ciertos artículos se incluyen con la finalidad de comparar y validar la información que se obtengan en el transcurso de la investigación.

Por último, también se desarrolla una revisión sobre las herramientas empleadas para la experimentación de este trabajo, con el objetivo de explicar el uso correcto de los recursos para formar una estructura adecuada de escenarios experimentales en la metodología.

2.2. Teorema de la codificación de canal ruidoso.

Uno de los fundamentos más importantes de la teoría que sustenta los sistemas de comunicaciones modernos corresponde al Teorema de la Codificación de Canal ruidoso (Noisy-Channel Coding Theorem), o también conocido como el límite de Shannon [12]. Este teorema describe el proceso matemático de la transmisión de información y establece que es posible transmitir información sobre un canal ruidoso con una probabilidad de error tendiente a cero, siempre que la tasa de transmisión de dicha información esté por debajo de cierto límite, al que Shannon denominó la capacidad del canal, y la longitud de las palabras codificadas tienda al infinito. Por su parte Kolmogorov [26], presentó una base lógica para explicar la Teoría de la información desde la perspectiva de la complejidad computacional y explicar los conceptos de Entropía e Información Mutua.

Un canal es un medio físico mediante el cual se transmite información. Generalmente, debido a sus condiciones físicas, las señales transmitidas sobre un canal experimentan diferentes distorsiones y cambios, que convencionalmente, se entienden como ruido que el canal añade a las señales. Generalmente el modelado de los canales se realiza siguiendo un modelo probabilístico que, en medida de lo posible, imitará el comportamiento real o esperado del canal [27]. Este modelado por lo general corresponde a una caracterización del ruido añadido por el canal como una variable aleatoria con una determinada distribución. Es así que surge la pregunta ¿Cómo es posible enviar información y que arribe correctamente a su destino?. Para esta pregunta existen dos respuestas ampliamente aceptadas: la primera corresponde al esquema basado en el uso de Automatic Repeat reQuest (ARQ) de aquellos mensajes que fueron recibidos erróneamente, mientras que la segunda, corresponde a uso de técnicas Forward Error Correction (FEC).

El primer esquema consiste en detectar si la palabra recibida contiene uno o más errores con el objetivo de solicitar al transmisor la repetición de los mensajes erróneos. En contraste, los esquemas de corrección de errores hacia adelante se fundamentan en generar un código lo suficientemente robusto y que contenga las características que le permitan la corrección de errores a través de un algoritmo de decodificación [27]. Estas dos soluciones pueden ser usadas de forma conjunta en esquemas híbridos FEC/ARQ, puesto que se puede generar códigos capaces de controlar y corregir un cierto número de errores. Sin embargo, en caso de que el decodificador no logre corregir los errores generados por el canal, se solicitará una retransmisión.

Los esquemas de corrección de errores hacia adelante se fundamentan en el uso de codificadores y decodificadores de canal. Un codificador se alimenta de los mensajes de entrada (usualmente representados en código binario plano de m bits) y los transforman en una secuencia distinta y de mayor longitud, en la que se cuenta con n bits donde se incluye información adicional que le da robustez, y que finalmente es explotada por el decodificador para estimar los mensajes transmitidos con mayor precisión dados los efectos adversos introducidos por el canal en forma de ruido, distorsión o interferencia.

Un factor muy importante al momento de construir o utilizar un codificador es la tasa

de código (denotado como *rate*), y corresponde a la relación entre el número de bits que ingresan al codificador y el número de bits ya codificados que salen del mismo. Teniendo en cuenta el hecho de que el trabajo del codificador es agregar redundancia al mensaje original, el mensaje codificado tiene una extensión mayor al original y por esto la tasa de código se mantendrá siempre entre 0 y 1.

Este trabajo se centra en la exploración y descubrimiento de nuevos códigos sobre un modelo de canal Binary Symmetric Channel (BSC) 2.2. Este es el modelo más básico de un canal binario sin memoria. En un BSC, el transmisor envía los bits 0 y 1. En este modelo de canal existe una probabilidad q de que el bit enviado se "voltee", es decir, que el receptor obtenga lo opuesto a lo enviado (un cero transmitido es recibido como un uno y viceversa) y una probabilidad $p = 1 - q$ de que el bit enviado sea el mismo que se ha recibido. En caso de que $p > q$ la probabilidad de que el receptor reciba un bloque sin errores es mayor a la probabilidad de recibir un bloque con un solo error y así mismo la probabilidad de recibir una secuencia con un error es mayor a la probabilidad de recibir un bloque con 2 o más errores [28].

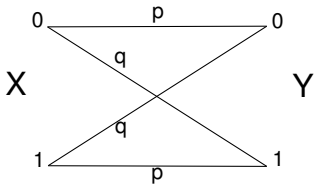

Figura 2.1: BSC con probabilidad de error q

La Figura 2.1 muestra un BSC cuya entrada está denotada por X y su salida por Y. En [27] se presentan los criterios de diseño y medidas de desempeño de los sistemas de codificación para un canal BSC, donde se indica claramente que las probabilidades de transición del canal, denotadas por $P(y_i|x_i)$ para el i-ésimo uso del canal, están dadas por:

$$P(y_i = 1|x_i = 0) = P(y_i = 0|x_i = 1) = q, \tag{2.1}$$

$$P(y_i = 1|x_i = 1) = P(y_i = 0|x_i = 0) = 1 - q \tag{2.2}$$

En estas ecuaciones q es la probabilidad de cruce del canal (channel flipping probability). Como consecuencia de la característica de falta de memoria del canal BSC

tenemos que:

$$P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|x_i), \tag{2.3}$$

donde $\mathbf{y}$ corresponde a $[y_1, y_2, y_3, ...]$ y $\mathbf{x}$ es igual a $[x_1, x_2, x_3, ...]$.

Para el diseño de un decodificador se considera el criterio de minimizar la probabilidad de fallo al decodificar la palabra de código recibida, lo que se traduce en maximizar la probabilidad a posteriori para el canal, de esa forma la decisión óptima para un canal BSC está dada por:

$$\hat{v} = \arg\max_v P(\mathbf{y}|\mathbf{x}) = \arg\max_v \frac{P(\mathbf{y}|\mathbf{x})P(\mathbf{x})}{P(\mathbf{y})} \tag{2.4}$$

donde $\arg\max_v f(v)$ es el argumento v que maximiza la función $f(v)$. De esta manera, se decodifica el mensaje v que maximiza la probabilidad a posteriori.

De forma frecuente, las palabras que ingresan al canal tienen la misma probabilidad de ocurrir, es por esto que $P(\mathbf{x})$ es independiente de $\mathbf{x}$ y de v. Debido a que $P(\mathbf{y})$ también es independiente de v, se puede reemplazar la regla de máxima probabilidad a posteriori por la regla de máxima verosimilitud [27].

$$\hat{v} = \arg\max_v P(\mathbf{y}|\mathbf{x}). \tag{2.5}$$

Utilizando la monotonía de la función logarítmica y (2.3) se tiene que para el canal binario simétrico:

$$\hat{v} = \arg\max_v \log \prod_i P(y_i|x_i) \tag{2.6}$$

$$= \arg\max_v \sum_i \log P(y_i|x_i) \tag{2.7}$$

$$= \arg\max_v [d_H(y,x)\log(q) + (n - d_H(y,x))\log(1-q)] \tag{2.8}$$

$$= \arg\max_v [d_H(y,x)\log\left(\frac{q}{1-q}\right) + n\log(1-q)] \tag{2.9}$$

Y dado que $\log\left[\frac{q}{1-q}\right] < 0$ y $n\log(1-q)$ no son funciones de v, se tiene que:

$$\hat{v} = \arg\min_v d_H(\mathbf{y}, \mathbf{x}) \tag{2.10}$$

donde n es la longitud de la palabra de código y d_H corresponde a la distancia de Hamming entre las secuencias $\mathbf{x}$ y $\mathbf{y}$.

De esta forma se entiende que el criterio para el diseño de un decodificador funcional al canal BSC corresponde al de la regla de máxima verosimilitud, es decir que el decodificador escoja una palabra de código lo más cercana posible a la salida del canal y en términos de distancia de Hamming. Así mismo se comprende la necesidad de diseñar códigos que maximicen la distancia de Hamming mínima entre dos palabras de código. En la práctica, el problema de diseñar nuevos códigos radica en que hallar las palabras del código que minimicen la distancia de Hamming es un proceso sumamente complejo excepto para códigos muy simples como los códigos de Hamming [27].

Para medir el rendimiento de los códigos diseñados se utiliza la probabilidad de error de bit P_b, que corresponde a la probabilidad de que la decisión ejecutada por el decodificador para el bit i-ésimo $\hat{u}_i$ sea distinta al bit de entrada al codificador u_i.

$$P_b = Pr\{\hat{u}_i \neq u_i\} \tag{2.11}$$

Comúnmente a P_b también se le conoce como Bit Error Rate (BER) o tasa de error de bit.

Adicionalmente, en el contexto de la codificación de canal, es fundamental referirse al concepto de su capacidad. Desde el año 1948 en el que Claude Shannon publicó su artículo sobre la teoría de la información [12] hasta el inicio de la década de los 90 era una idea generalizada que la única forma de operar cerca de la capacidad de los canales era con el uso de códigos extensos y con una velocidad de codificación y decodificación tan lenta que se volvían imprácticos. Sin embargo, como se detalla en “Elements of Information Theory” de Thomas Cover [29], el desarrollo de Turbo códigos en la década de 1990 [14] demostró que se puede alcanzar un rendimiento cercano a la capacidad del canal en la práctica. La capacidad de un BSC está dada por [27]:

$$C_{BSC} = 1 - H(q), \tag{2.12}$$

donde $H(q)$ representa la función de entropía binaria para una probabilidad q.

2.3. Técnicas de codificación convencionales

Desde el nacimiento de la teoría de la información gracias al trabajo del matemático e ingeniero estadounidense Claude Shannon [12], se han buscado de forma exhaustiva códigos de canal eficientes que permitan transmitir información lo más cercano posible a la capacidad del canal en cuestión. Los esfuerzos en el desarrollo de códigos prácticos capaces de acercarse o incluso alcanzar los límites de comunicación teóricos, se suman al trabajo de Shannon y de múltiples investigadores entre los que se destacan Robert Gallager [30], Daniel Costello [31], Robert McEliece [32], Andrey Kolmogorov [26], Tomas Cover [29]. Esto se consigue al generar codificadores que añadan la justa y necesaria cantidad de redundancia al código y decodificadores que exploten de la mejor forma posible dicha redundancia, de esa forma minimizando la cantidad de errores en la transmisión tanto como sea posible. El resultado del esfuerzo de varios matemáticos e ingenieros tuvo sus respectivos frutos con la creación de distintos códigos, entre los más relevantes, se puede mencionar a los códigos convolucionales, de Hamming, Turbo códigos, códigos polares, Low-Density Parity-Check (LDPC), etc. Estos códigos facilitaron una amplia evolución en la transmisión de información ya que disminuyeron las tasas de errores de manera significativa en las comunicaciones en comparación a las transmisiones de datos sin codificar.

Evidentemente en la práctica, la forma de determinar el desempeño de los distintos códigos descubiertos y desarrollados así como sus codificadores y decodificadores es hacer pruebas respectivas en canales ya sean estos simulados o reales. Al hacer pruebas en canales simulados se tiene una astronómica ventaja y es que es posible controlar prácticamente la totalidad de las variables lo que permite determinar de mejor forma el desempeño de los códigos que se prueban. En [1] se presenta una evaluación sobre el uso de códigos de Hamming en un canal caracterizado por ruido aditivo Gaussiano Additive White Gaussian Noise (AWGN).

En particular, la lógica detrás de los códigos de Hamming según lo visto en [33] es tomar una matriz cuadrada y que su dimensión sea potencia de dos, en [34] se usa una matriz cuadrada de dimensión igual a cuatro, es decir de 16 valores. Esta matriz es utilizada para transmitir 11 bits ya que el resto de posiciones serán rellenadas con

bits de paridad. La matriz incluyendo únicamente los bits de transmisión se muestra en la Figura 2.2:

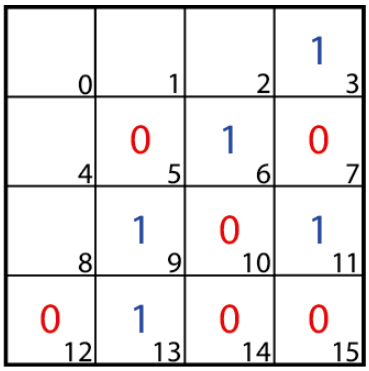

Figura 2.2: Matriz de Hamming con bits de información.

La matriz se divide en 4 partes cada una de ellas contempla una porción especifica tal como se puede ver en la Figura 2.3.

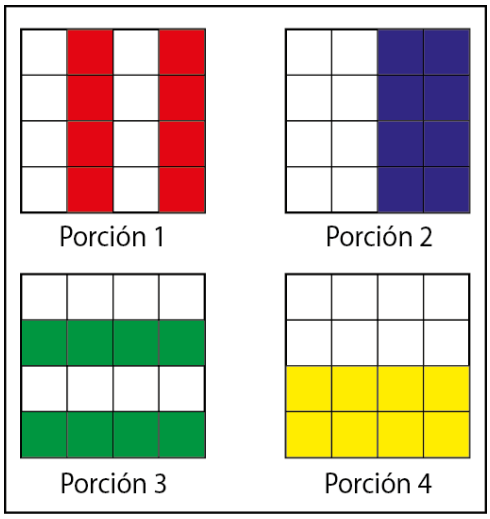

Figura 2.3: Porciones de la matriz para bits de paridad.

Tal como se puede ver en 2.3, las porciones se encuentran divididas de la siguiente forma:

- Porción 1 : Columnas pares
- Porción 2 : Columnas del fondo (izquierda)
- Porción 3 : Filas pares
- Porción 4 : Filas del fondo

La lógica para incorporar los bits de paridad dentro de sus respectivas posiciones es la siguiente: en cada porción se cuenta la cantidad de bits activos, si estos son pares, el bit de paridad sera 0, por otro lado, si estos son impares, el bit de paridad sera 1, manteniendo así la paridad de bits activos en cada porción. Este proceso resulta en el contenido de la Figura 2.4.

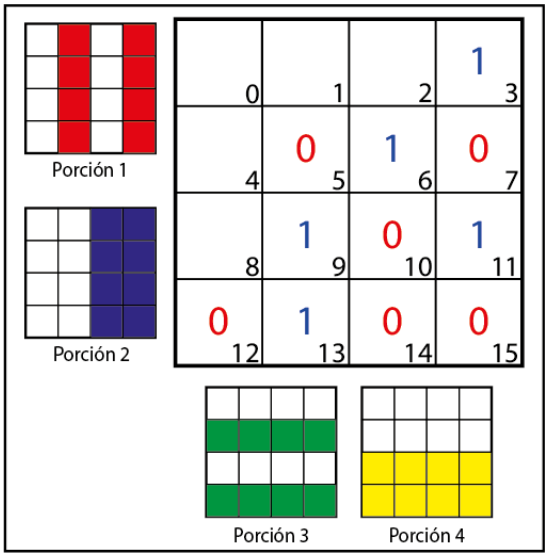

Figura 2.4: Matriz de codificada con bits de paridad de Hamming.

Para poder detectar un error basta con tomar una a una las porciones y revisar si la cantidad de bits en 1 corresponde al bit de paridad. En caso de no ser así se podrá con cuatro sencillas preguntas rastrear la posición del error, las “preguntas” consistirían en determinar si en cada porción de la matriz el bit de paridad corresponde a su número de bits en 1. Si tomamos las respuestas de tal forma que “SI” sea equivalente a 1 y “NO” sea equivalente a 0 entonces al colocar las cuatro respuestas juntas se nos devuelve (en binario) la posición en la que se generó el error, haciendo así fácil corregirlo. Esto se puede ver de mejor forma en la Figura 2.5.

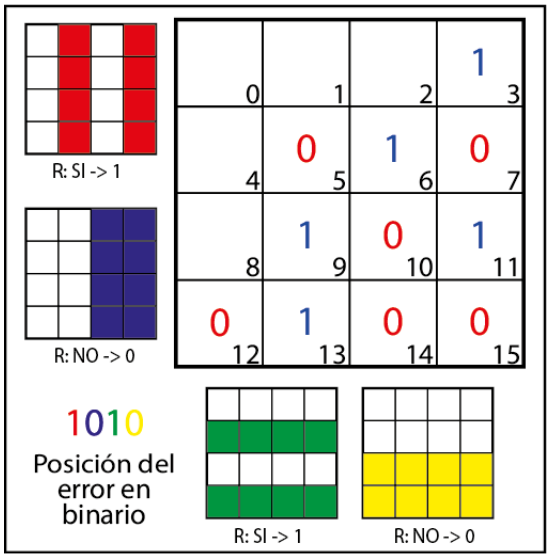

Figura 2.5: Matriz de Hamming con error detectado.

Como se muestra en la Figura 2.5, se ha encontrado un error, y detectado su ubicación gracias a los bits de paridad, por lo que es posible corregirlo. El punto débil de los códigos de Hamming con bit de paridad es el hecho de que pueden corregir únicamente un solo error.

En [1] se ha propuesto un método para analizar la probabilidad de error de bit de un código de Hamming cuando los símbolos tienen distintos niveles de energía, pero además se ha simulado sobre un canal AWGN de modo que se pueda visualizar la probabilidad de error al simular las condiciones de un canal. La Figura 2.6 compara el BER al usar datos crudos y datos codificados con Hamming (7,4). Esta notación indica que se envían 7 bits de los cuales únicamente 4 son bits de información, por lo que los 3 bits restantes corresponden a bits de codificación.

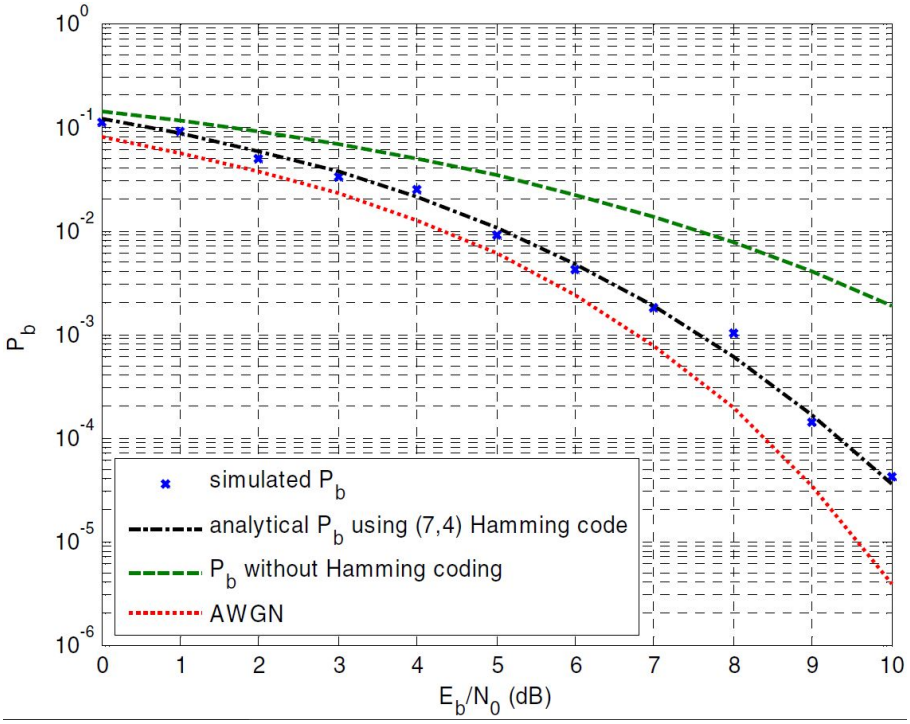

Figura 2.6: Probabilidad de error analítica y simulada con códigos Hamming [1], © 2005 IEEE.

Claramente en la Figura 2.6 se puede ver que la diferencia entre usar códigos de Hamming y enviar datos “crudos” o sin codificar es bastante grande, puesto que el BER se reduce significativamente, como se puede apreciar para un Signal-to-Noise Rati (SNR) **(**$E_b/N0$**)** de 10 dB, donde se observa una diferencia de varios ordenes de magnitud en el BER.

Siguiendo esta metodología, es posible observar el desempeño de otros códigos, como son los códigos polares y los turbo códigos.

Los turbo códigos, como se documenta en [3], corresponden a un tipo de códigos que se construyen uniendo dos codificadores convolucionales mediante un intercalador de bits. El intercalador de bits reorganiza la secuencia de bits antes de que estos pasen al segundo codificador para así aumentar la dispersión de los errores y mejorar la capacidad de corrección. Generalmente, los codificadores turbo tienen una tasa de bit (rate) de 1/3. Para que esto sea posible, se usan dos codificadores convolucionales con tasa de 1/2 en paralelo. La decodificación de estos códigos utiliza de forma iterativa el Bahl, Cocke, Jelinek, and Raviv (Algoritmo) (BCJR), el cual calcula las probabilidades a posteriori de cada bit de información dada toda la secuencia recibida.

La decodificación se realiza mediante dos decodificadores BCJR: el primero procesa los bits recibidos en primera instancia y genera información de fiabilidad para cada bit de información. Esta información pasa al segundo decodificador BCJR, que procesa los bits reorganizados por el intercalador. La información extrínseca obtenida del segundo decodificador se retroalimenta al primero. Finalmente, este intercambio de información continúa hasta que las probabilidades a posteriori estimadas por ambos decodificadores sean consistentes.

Por otro lado, los códigos polares, como se describe en [18], son un tipo de códigos que aprovechan la polarización de canales para lograr un desempeño superior. En la etapa de codificación, se utiliza una matriz de polarización construida mediante operaciones que dividen los canales en canales polarizados fuertes y débiles. Los canales polarizados fuertes son aquellos extremadamente confiables, con una probabilidad de error muy baja, mientras que los canales polarizados débiles son poco confiables, con una alta probabilidad de error. De esta forma, se puede enviar la información a través de los canales fuertes y los bits de paridad a través de los canales débiles. Al hacer esto, se crea un patrón donde los bits de información son más propensos a ser recibidos con mayor precisión. En la decodificación, se utiliza un algoritmo conocido como algoritmo de decodificación suave de polaridad sucesiva, que aprovecha la estructura jerárquica de los canales polarizados para realizar una decodificación eficiente y de baja complejidad. A grandes rasgos, este algoritmo estima la secuencia de bits transmitida usando la información recibida a través del canal y la estructura polarizada de los canales.

En el artículo escrito por Mahdi Bersali, Hocine Ait-Saadi y Messaoud Bensebti se documenta el rendimiento de los códigos polares en términos de la tasa de error de bit para distintas longitudes de código y tasas de codificación. Además, se explica que el algoritmo usado para construir los códigos polares se basa en una selección de los mejores canales de bits según un valor inicial de SNR llamado SNR de diseño. Adicionalmente usando como métrica el BER, se comparan los turbo códigos y los códigos polares al pasar estos sobre un canal de ruido Gaussiano blanco aditivo binario [2]. La Figura 2.7, tomada de [2] muestra dos gráficas: a la izquierda se pueden observar los resultados al usar distintos niveles de SNR de diseño, mientras que a la derecha

se pueden ver los resultados obtenidos en [2] al utilizar códigos polares con distintas tasas de código. Finalmente en 2.8 podemos ver la comparación de desempeño de turbo códigos y códigos polares sobre el mismo tipo de canal.

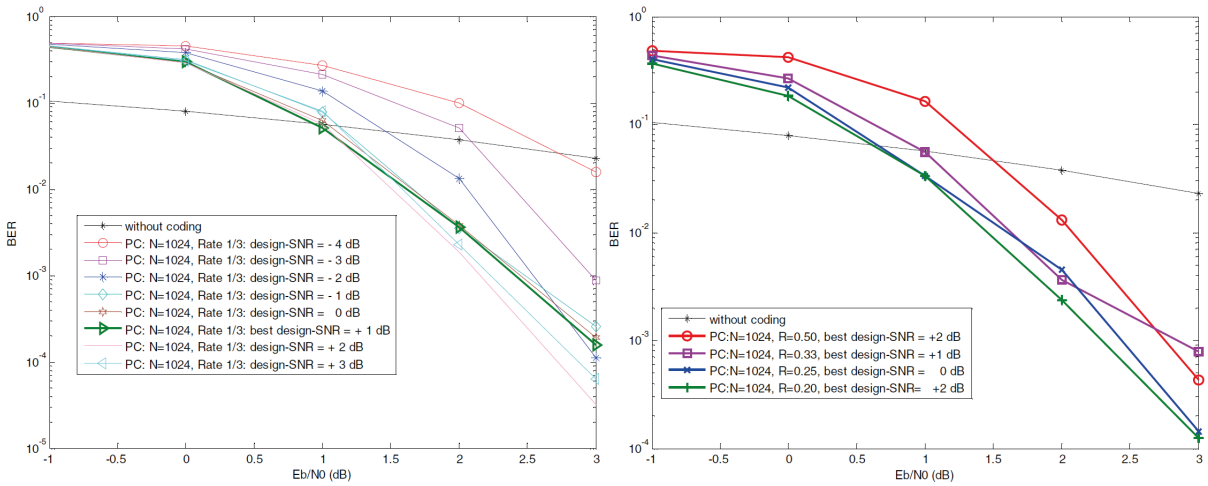

Figura 2.7: Izquierda: Resultado de códigos polares con distintos niveles de SNR de diseño. Derecha: Resultado de códigos polares con distintas tasas de código [2], © 2017 IEEE.

Evidentemente, la Figura 2.7 refleja el hecho de que utilizar una técnica de codificación es siempre mejor que no utilizarla. Además podemos ver en la parte izquierda las diferencias de desempeño según el nivel de SNR de diseño utilizado, esto es interesante ya que permite observar como al tener un mayor nivel de SNR de diseño, el rendimiento de los códigos diseñados serán mejores. Además, tal como se indica en [2], en la parte derecha de la Figura 2.7 se puede ver como una realidad teórica se cumple, esta es que para una longitud de código determinada, se nota una mejora en los valores de BER a medida que se disminuye la tasa de codificación. Como era de esperarse, a medida que se incrementa el nivel de SNR de diseño, el rendimiento de los códigos mejora. Sin embargo, en los niveles más altos de SNR de la simulación, se observa que el código generado con 2 dB tiene un mejor rendimiento que el código generado con 3 dB. Dado que esto no se explica en [2], podemos concluir que podría deberse a la naturaleza de su diseño. Los códigos diseñados para 2 dB están optimizados para funcionar mejor en condiciones de canal adversas, es decir, con más ruido, lo que les permite manejar una mayor cantidad de errores. Por lo tanto, cuando la calidad del canal mejora, estos códigos pueden mejorar aún más su desempeño

debido a su robustez inherente, incluso superando a aquellos códigos diseñados con un SNR mayor. Por otro lado, en la imagen de la derecha es claro notar como mientras se disminuye la tasa de codificación los resultados de BER son mejores. Estos resultados evidencian que mientras menor es la tasa de codificación, mayor es la cantidad de bits de redundancia que se envían por cada bit de información lo que permite una tolerancia mayor a errores.

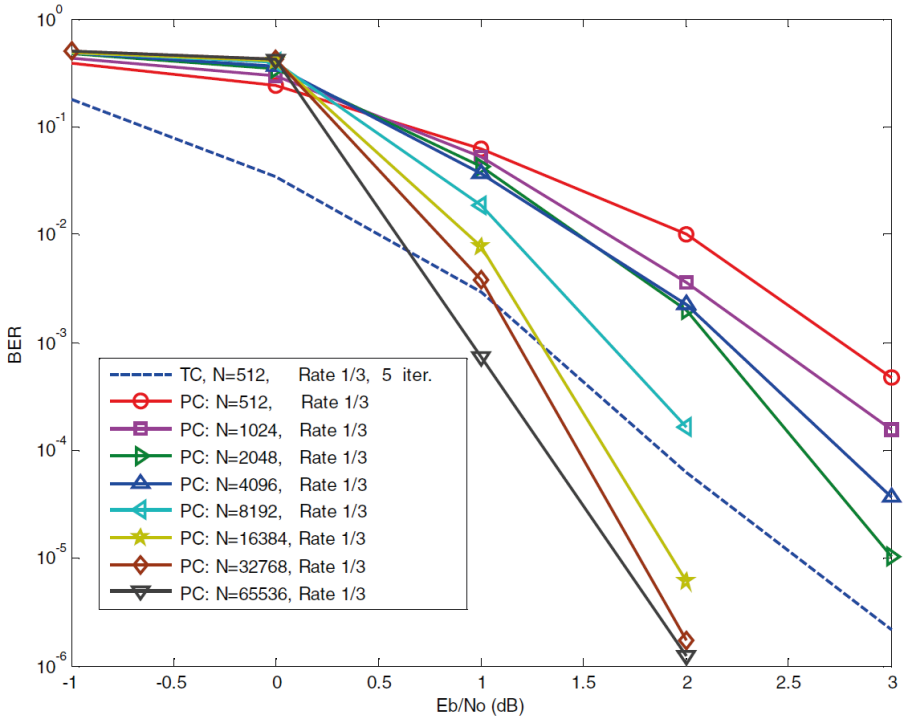

Figura 2.8: Comparación del desempeño de turbo códigos y códigos polares [2], © 2005 IEEE.

En la Figura 2.8 los resultados son bastante claros: en caso de utilizar turbo códigos con longitud de 512, tenemos mejores resultados que los códigos polares con la excepción de los códigos polares que utilizan longitudes de código de 16384, 32768, 65536. No obstante, aquí, tal como se menciona en [2], podemos ver otro hecho teórico, el cual indica que al incrementar la longitud de código tendremos códigos polares con mejor desempeño.

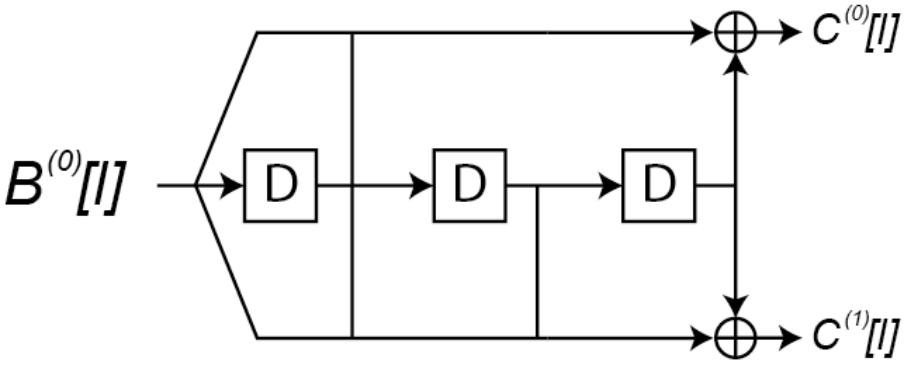

Figura 2.9: Codificador convolucional, imagen adaptada de [3].

La Figura 2.9 muestra un codificador convolucional con tasa de codificación (rate) 1/2. Estos codificadores funcionan de la siguiente forma: la información (B) ingresa al codificador por la izquierda del diagrama; los bits componentes del mensaje a codificar ingresar de uno en uno, y se desplazan progresivamente de forma que se coloca una cierta cantidad de bits (definida por el kernel del codificador) dentro del codificador a la vez. Estos bits presentan retrasos entre sí, representados por los bloques etiquetados como D, que corresponden a una unidad de retardo. En cada instante de tiempo (instante en el que los bits toman su posición en el codificador), se realiza la suma en módulo 2 o Exclusive OR (XOR) de los elementos indicados. En el ejemplo presentado en 2.9, el primer bit de codificación es el resultado de la suma de las dos primeras posiciones en módulo dos y el segundo bit de codificación es el resultado de la suma de las tres posiciones en módulo dos. Los codificadores convolucionales dependen de parámetros como la tasa de codificación o la memoria total del código.

La decodificación de los códigos convolucionales se realiza utilizando el algoritmo de Viterbi [35]. Este algoritmo implica analizar toda la secuencia de bits recibida para encontrar la secuencia que minimice la distancia de Hamming con respecto a las posibles secuencias generadas por el codificador. En este algoritmo se asume que los estados iniciales y finales del codificador son conocidos, y se utiliza una métrica de rama para evaluar la probabilidad de cada transición entre estados. El proceso de decodificación selecciona de forma iterativa el camino más probable, acumulando métricas de rama para determinar la secuencia más probable de bits de entrada que coincide con la secuencia recibida. Al ejecutar este proceso, se corrigen los posibles errores introducidos por el canal.

En [36] se hace una comparación del rendimiento que muestran los códigos convolucionales y Random Codes Based on Quasigroups (RCBQ) en un canal binario simétrico. Por un lado tenemos a los códigos convolucionales, cuyo desempeño se documenta en [16] y corresponden a uno de los primeros y más utilizados códigos para corrección de errores en la práctica. Por otro lado, los códigos aleatorios basados en cuasigrupos, se documentan en [37], y corresponden a una clase de códigos de corrección de errores generados por transformaciones de cadenas de cuasigrupos casi aleatorios que pueden ser decodificados de forma eficiente con esquemas iterativos.

En [36] el proceso de decodificación de los códigos convolucionales usa el algoritmo de Viterbi, el cual es prácticamente un estándar para estos códigos. El proceso de decodificación utilizado para los códigos aleatorios basados en cuasigrupos es el algoritmo de decodificación por cortes [38]. Finalmente los experimentos reportados en [36] son:

Para RCBQ con el algoritmo de Decodificación por Cortes se realizaron para el código (72, 288) utilizando los siguientes parámetros del código:

- patrón de redundancia: 1100 1110 1100 1100 1110 1100 1100 1100 0000
- dos claves diferentes: k1 = 01234 y k2 = 56789
- cuasigrupo de orden 16

Y para los códigos convolucionales se realizaron usando la estructura trellis y los siguientes parámetros:

- bits de datos k = 1, para cada bit entrante, hay n bits
- palabras de código de n bits, n = 4, cada bit se codifica con 4 bits
- longitud de restricción K = 3, el número de estados es 4 $(2^K - 1)$

Para todos los experimentos se utilizó un canal binario simétrico con distintas probabilidades para error de bit. Los resultados de los dos códigos evaluados se presentan en la Figura 2 de [36], donde se aprecia cómo los códigos convolucionales exhiben un menor nivel de tasa de error de bits mientras el canal tiene una probabilidad de error de canal mayor a 0.07. En contraste, los códigos aleatorios basados en cuasigrupos

tienen un desempeño claramente mejor que los convolucionales cuando se mejora la calidad del canal BSC. Nótese que cuando la probabilidad de error del canal es aproximadamente 0.07 los dos códigos tienen un desempeño similar, sin embargo a medida que la probabilidad de error disminuye, los códigos aleatorios basados en cuasigrupos mejoran de forma notable en comparación con los códigos convolucionales.

Adicionalmente, [4] presenta un tipo de códigos recientemente desarrollados con la peculiaridad de ser rateless", los códigos espinales (Spinal codes). Estos códigos introducen una función de tipo *hash* como el núcleo de su proceso de codificación, de esa manera pueden generar bits pseudoaleatorios de forma continua. Básicamente el proceso de codificación incluye cuatro pasos clave:

1. Un mensaje de n bits, denotado por M, se divide en segmentos de k bits.
2. El codificador usa una función *hash* para mapear el segmento del mensaje a un estado de bits.
3. El estado de bits se usa para inicializar un Random Number Generator (RNG) para generar una secuencia de símbolos pseudoaleatorios de c bits.
4. El emisor mapea los símbolos de c bits a un conjunto de entradas del canal para ajustarse a las características del canal.

En particular [4] se enfoca en el estudio del límite asintótico ajustado de la tasa de error de bits de los códigos espinales en un canal binario simétrico, lo que constituye un paso fundamental para comprender su comportamiento y optimizar su desempeño. Utilizando este análisis como base, se desarrolla una estrategia de perforación óptima para mejorar la eficiencia de los códigos espinales en dicho canal. Esta estrategia de perforación se diseña meticulosamente para dirigir más símbolos hacia las regiones del código que son más propensas a errores, con el fin de garantizar una decodificación exitosa. Este enfoque estratégico no solo reduce la tasa de error de bits, sino que también mejora significativamente el rendimiento del BER global del sistema. Para validar la efectividad y la robustez de esta estrategia de perforación propuesta, se llevan a cabo simulaciones exhaustivas en diversas condiciones de canal, proporcionando así una evaluación de su desempeño. En la Figura 2.10 se presentan los resultados de la simulación de códigos espinales originales y códigos espinales con perforación

óptima, y un *rate* de 1/3.

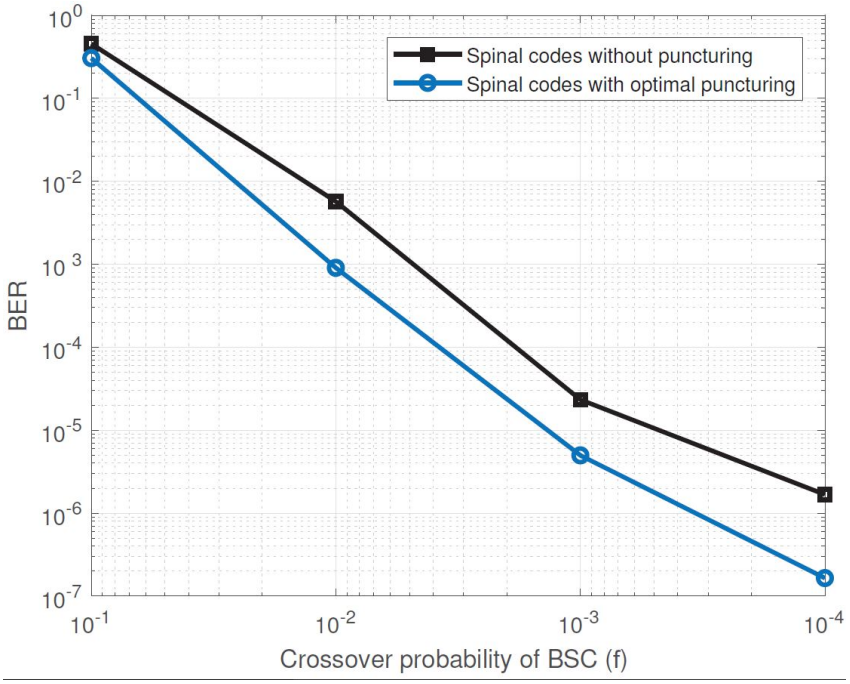

Figura 2.10: Desempeño de códigos espinales con y sin perforación [4], © 2020 IEEE.

Puede observarse que, de manera similar a los demás códigos mencionados anteriormente, los códigos espinales muestran una mejora sustancial en la tasa de error de bit. El estudio detallado en [4] es de gran relevancia, ya que logra un avance significativo en el rendimiento de estos códigos, mejorando su eficacia en casi un orden de magnitud con respecto a los códigos espinales originales. Esto demuestra de manera concluyente que la estrategia de perforación supera con creces a los códigos espinales originales, lo que tiene implicaciones importantes en el diseño y la implementación de sistemas de comunicación.

En la misma linea, se puede mencionar que los códigos polares, desarrollados por Erdal Arikan [18] pueden alcanzar la capacidad de Shannon de la clase de canales binarios discretos sin memoria con una complejidad de codificación y decodificación bastante baja. Estos códigos usan la polarización de canal la cual establece que al momento de combinar y dividir N copias independientes de canales Binary Discrete Memoryless Channel (B-DMC), los canales se polarizan, es decir, se tienen canales libres de ruido. Una evaluación del rendimiento de códigos polares en canales binarios simétricos se muestra en [5]. Esta imagen muestra distintos tama-

ños de bloque en canales AWGN y en canales BSC. El artículo [5] incluye detalles importantes sobre la implementación de estos códigos sobre un canal binario simétrico, en la que se definen los alfabetos de entrada y salida del canal como $X = 0, 1$ e $Y = 0, 1$ respectivamente, un bit transmitido se invierte con una probabilidad q. Las probabilidades de transición son $P(Y = 0|X = 1) = P(Y = 1|X = 0) = q$ y $P(Y = 0|X = 0) = P(Y = 1|X = 1) = 1 - q$. La tasa de errores de bit usando códigos polares en canales BSC se obtiene mediante simulaciones en Matlab según los siguientes pasos:

1. Se generan K bits aleatorios.
2. Esos bits se introducen como entrada al codificador que produce N bits.
3. Luego se agrega ruido binario. Se calculan los Log-Likelihood Ratio (LLR) de los resultados.
4. Estos LLR se introducen en el decodificador Successive Cancellation (SC).
5. El decodificador SC produce una estimación de los datos de entrada.
6. Al final se comparan los bits de salida con los bits de entrada y se cuentan los errores.

Con estos pasos se ha comparado también el desempeño entre los códigos polares de distinta longitud de bloque (N), estos bloques tienen una tasa de código de $1/2(K/N)$, cuyos resultados se muestran en La Figura 2.11.

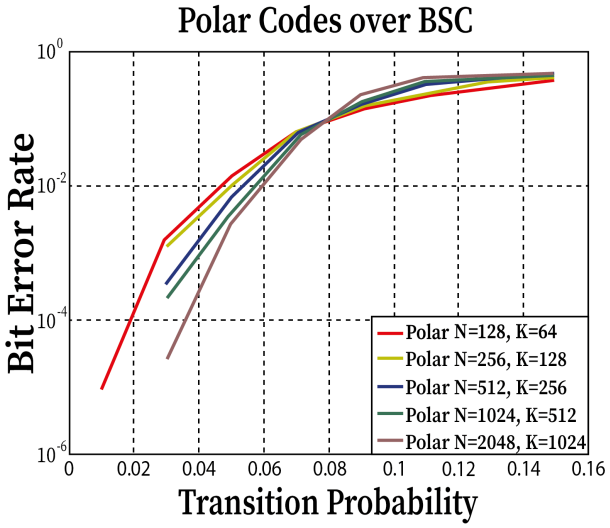

Figura 2.11: Desempeño de códigos polares en canal BSC [5].

En La Figura 2.11 se evidencia de forma clara como a medida que aumenta el tamaño de bloque (N), se tiene un mejor desempeño del código, lo que se traduce en un menor valor BER. Al igual que en otros códigos convencionales, cuando se tiene un canal en muy malas condiciones (q = 0.1), la cantidad de errores que se puede corregir es muy pequeña y por tanto, en estos casos el BER es significativamente alto. No obstante, a medida de que el canal mejora (q se reduce) la cantidad de errores se corrige crece de manera veloz, es por esto que en La Figura 2.11 se observa una curva creciente.

2.4. Técnicas de Codificación de Canal basada en Redes Neuronales

El constante desarrollo y mejoras que se presentan en relación a la capacidad computacional y las continuas invenciones cada vez más eficientes de algoritmos de Inteligencia Artificial (IA) y Aprendizaje Automático (AA) tal como se menciona en [39], han conseguido que actualmente la IA sea empleada en una amplia gama de aplicaciones, entre ellas se encuentra el procesamiento de imágenes, clasificación de información, ciencia de datos, en la industria y el que nos concierne, las comunicaciones digitales, entre otros.

La razón fundamental para hacer uso de técnicas de codificación basadas en redes neuronales, es evidentemente la solución que ha representado el uso de Artificial Neural Network (ANN) para varios problemas de la actualidad. Por ello, apoyarnos de esta tecnología para generar nuevos desarrollos en el área de las comunicaciones digitales, específicamente centrando la investigación en la codificación de canal es un paso completamente razonable en la búsqueda de obtener métodos de codificación óptimos que puedan acercarse a la capacidad del canal, manteniendo una complejidad de implementación baja, esto teniendo en cuenta el arduo trabajo y la dificultad que tiene este nicho de estudio.

2.4.1. Conceptos y funcionamiento general de redes neuronales artificiales

Esta Sección presenta una breve revisión sobre los principios fundamentales que rigen los modelos de ANN. Según [6], en el modelo computacional de las redes de neuronas artificiales, la gran diferencia con los programas convencionales de compu-

tadora, es que estas redes generan la información de entrada para obtener una salida o respuesta, en lugar de ejecutar un algoritmo predefinido. El proceso de elaboración es dependiente de las características estructurales y funcionales de la red neuronal.

Estas redes poseen elementos de procesamiento que tratan de simular las células del cerebro humano, y que usualmente, se conectan con otras formando diferentes capas o niveles de procesamiento. Las neuronas artificiales poseen un estado interno denominado nivel de activación y reciben señales las cuales inducen un cambio de estado. Se pueden representar con un conjunto de estados posibles, tal como 0 y 1 para estados inactivo y activo respectivamente, o también con un intervalo continuo de valores. Cada una de las neuronas que componen el modelo posee una función de transición de estado o función de activación que determina su nivel de activación a partir de las señales recibidas. Esta función puede tener un comportamiento lineal, umbral o cualquier otra función previamente definida. En La Figura 2.12, se muestra un modelo que representa la idea de una neurona artificial.

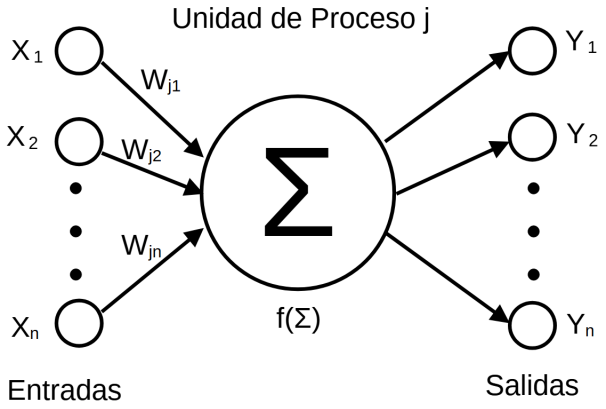

Figura 2.12: Modelo de neurona artificial, imagen adaptada de [6].

En La Figura 2.12, se observa que un grupo de entradas ingresan a la neurona artificial. Estas entradas se pueden definir mediante un vector $\overline{X}$, que análogamente corresponde a las señales de la sinapsis de una neurona biológica. Cada una de las señales se multiplica por un factor o peso correspondiente w_1, w_2, ..., w_n, para formar una combinación lineal. Cada uno de los pesos se asemeja a una conexión sináptica, en otras palabras, representa el nivel de concentración iónica de la sinapsis, y esto se representa por medio de un vector $\overline{W}$.

El elemento sumatorio corresponde al cuerpo de la neurona, la cual suma cada una

de las entradas ponderadas algebraicamente, dando como resultado una salida E, que no es más que una combinación lineal de las entradas y los pesos, como se muestra en (2.13).

$$E = x_1 w_1 + x_2 w_2 + ... + x_n w_n \tag{2.13}$$

La ecuación (2.13) se puede escribir de manera vectorial tal como se presenta en (2.14).

$$E = X^T W \tag{2.14}$$

Las señales de E, se atraviesan una función de activación f, generando una salida $\overline{Y}$, en base a la función de activación, se pueden generar diferentes tipos de modelos autómatas.

A la forma en que estas células artificiales se conectan entre sí se les denomina patrón de conectividad o arquitectura de la red. La estructura más básica de interconexión entre neuronas se le denomina la red multicapa, representada de forma gráfica en La Figura 2.13.

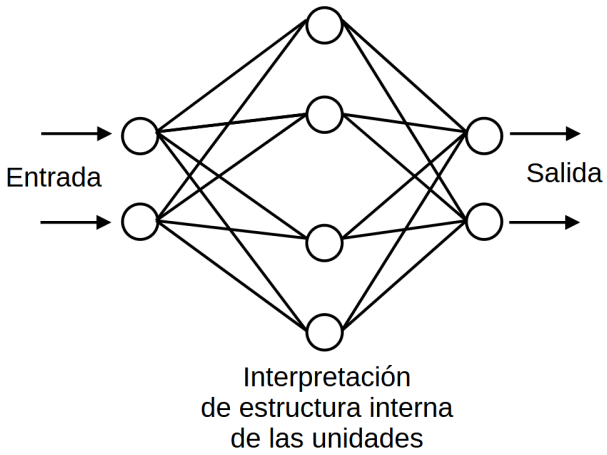

Figura 2.13: Esquema de una red de tres capas que están completamente conectadas, imagen adaptada de [6].

Esta estructura de implementación típica pertenece al paradigma conocido como “Feed-Foward”. Aquí, el primer nivel está constituido por las células de entrada, quie-

nes reciben los valores de patrones que están representados como vectores, y constituyen la entrada para la red. Después se incluye una serie de capas intermedias o capas ocultas, cuyas unidades de procesamiento responden a rasgos particulares que podrían aparecer en los patrones de entrada. Normalmente, pueden existir uno o más niveles ocultos. Luego, el último nivel se conoce como el nivel de salida. La salida de estas unidades en esencia es la salida de la red. Cada una de las interconexiones de las neuronas actúa como una ruta de comunicación, por medio de las cuales atraviesan valores numéricos. Cada valor es evaluado por los pesos de las conexiones y estos pesos son ajustados durante la fase de aprendizaje para lograr producir una red de neuronas artificial. Este ajuste de los pesos se realiza mediante un proceso de optimización en el que los errores producidos por la salida de la red (en comparación con los valores deseables) se utilizan hacia atrás, modificando los pesos de las conexiones de manera proporcional a dicho error (backpropagation).

Una red de neuronas artificial puede considerarse un grafo, que posee nodos que se encuentran constituido por unidades de proceso idénticas, las cuales propagan información por medio de los arcos (conexiones entre los nodos). En este grafo se distinguen tres diferentes tipos de nodos, los de entrada, los de salida e intermedios.

En función a esta estructura, se puede explicar brevemente el funcionamiento de la red neuronal. Cada uno de los vectores de entrada ingresan a la red copiando sus valores en las células de entrada correspondientes. Luego, cada célula al recibir todas sus entradas, procesa la información y crea una salida la cual es propagada por medio de las conexiones hacia las células destino (propagación hacia adelante o Feed-Foward). Después de que la entrada se propagó completamente por la red, se crea un vector de salida que está compuesto por múltiples valores de salida presente en la capa que contiene las células de salida. Este comportamiento se describe mediante (2.15).

$$\vec{S} = F\left(F\left(F\left(\vec{X} \cdot W_1\right) \cdot W_2\right)\right) \tag{2.15}$$

donde W_1 y W_2 corresponden a los pesos de las capas, F a la función de activación, $\vec{X}$ al vector de entrada y $\vec{S}$ corresponde al vector de salida.

Según se indica en [6], el aspecto más relevante de los modelos de ANN es el tipo de aprendizaje, pues este indica los problemas que la red es capaz de resolver y usualmente se basa en ejemplos. La capacidad de resolver problemas de la red es dependiente de que el conjunto de los ejemplos sea significativo y representativo. De esta manera, el proceso de aprendizaje implica el ajuste de los pesos de las conexiones por medio de la introducción de ejemplos y la verificación de si cumple con un criterio de convergencia. El proceso de aprendizaje puede ser supervisado, no supervisado o por refuerzo en función de la información que se posee. Para el aprendizaje supervisado, los ejemplos tienen datos y soluciones esperadas. En el no supervisado, la red tiene que identificar patrones de manera autónoma, mientras que para el aprendizaje por refuerzo se emplea evaluaciones generales de adecuación (o un mecanismo de recompensas) sin información precisa de errores.

El proceso de entrenamiento también debe realizarse con precaución, puesto que realizar un ajuste excesivo de datos de entrenamiento puede producir malas predicciones cuando nuevos datos se ingresan al modelo. Este fenómeno se denomina sobre ajuste y claramente disminuye la capacidad de generalización de la red. Para analizar si la red está produciendo salidas adecuadas, se realiza una división de los datos, donde se asigna una proporción de los mismos para entrenamiento y otra para validación y evaluación. Con el conjunto de datos de entrenamiento se realiza el ajuste de los pesos, mientras que el conjunto de validación se mide el error obtenido por el modelo. Si el error es bajo en el conjunto de validación, entonces, se garantiza una buena capacidad de generalización para cuando el modelo se pruebe con otro conjunto de datos de evaluación. Es importante recalcar que el conjunto de validación tiene que ser independiente, pero también cumplir con las mismas propiedades de un conjunto de entrenamiento. En la etapa de aprendizaje, hay pasos que resaltan y se pueden marcar como hitos en el proceso: en primer lugar se realiza una asignación de valores aleatorios a los pesos, que después se ajustan a medida que se va evaluando las salidas para cada uno de los patrones del conjunto de entrenamiento; acto seguido se evalúa el error con el conjunto de validación. Este proceso se repite hasta que el error caiga por debajo de un umbral determinado (o nivel de tolerancia del error) o se haya cumplido con cierto número de iteraciones. Si las condiciones se cumplen, entonces, el aprendizaje concluye y se obtiene el modelo de red resultante u óptima (aunque

siempre cabe la posibilidad de que no se alcance un mínimo global en la métrica de error evaluada).

2.4.2. Algoritmo de Retropropagación del gradiente

Este algoritmo, conocido en Inglés como *back-propagation*, corresponde a uno de los principios fundamentales que se aplican en el contexto del aprendizaje supervisado. En [40], se indica que este algoritmo nace de la crítica de Minsky al perceptrón de Rosenblatt [41], lo que produjo un estancamiento en las investigaciones de redes neuronales durante dos décadas, puesto que un perceptrón simple, no podía resolver el problema de una compuerta XOR. La solución a este problema requería de modelos de redes de más de una capa y un mecanismo para poder aprender (modificar los pesos de las conexiones) al interior de las capas internas. Entonces, el proceso de retro-propagación del gradiente surgió para permitir este aprendizaje.

El algoritmo de retro-propagación calcula los gradientes de la función de error empezando por las neuronas de la capa de salida, y los va propagando hacia atrás hasta llegar a la capa de entrada. Este algoritmo permite el entrenamiento de redes neuronales multicapas, solucionando ciertas limitaciones del perceptrón de una capa. De esta manera, se tiene un factor proporcional al error calculado en la salida y que puede utilizarse en cada capa para realizar la modificación de los pesos de las conexiones entre neuronas.

El algoritmo tiene algunas etapas que describimos a continuación: la primera es la propagación hacia adelante, en la que una entrada atraviesa la red neuronal capa por capa hasta llegar a la salida. Y en cada una de las neuronales se realiza un cálculo de la suma ponderada, esto significa que cada neurona recepta las salidas de las neuronas de la capa anterior, las multiplica por los pesos sinápticos y realiza la suma de estos valores junto con un sesgo (bias). Luego esta suma ponderada atraviesa una función de activación que puede ser, por ejemplo, un sigmoide, una función Rectified Linear Unit (ReLU), entre otras y con esto produce la salida de la neurona. El proceso se aplica para todas las neuronas en la capa de salida.

Cuando se obtienen las salidas, se realiza la comparación con los valores reales em-

pleando una función de pérdida, dicha función mide cuán lejos están las predicciones obtenidas por el modelo con respecto a los valores verdaderos, lo que constituye el principio fundamental del aprendizaje supervisado. Seguidamente, se realiza la retropropagación, con el objetivo de ajustar los pesos de la red minimizando el error calculado en la fase anterior. Para ello, el primer punto es calcular el gradiente de la pérdida en la capa de salida. Esto implica que para cada una de las neuronas de esta capa, se realiza el cálculo del gradiente de la función de pérdida con respecto a la salida. Esto se puede identificar en (2.16).

$$\delta_j = \frac{\partial E}{\partial s_j} \tag{2.16}$$

donde E corresponde a la función de pérdida y sj a la salida de la neurona. Posteriormente, se propaga el error hacia atrás por cada capa, para lo cual se emplea el valor del gradiente calculado en la capa de salida para calcular los valores de gradientes de capas anteriores. Esto se efectúa de forma iterativa desde la capa de salida hasta la capa de entrada. La ecuación (2.17), refleja matemáticamente este proceso.

$$\delta_j = \left(\sum \delta_k w_{jk}\right) \cdot f'\left(x_j\right) \tag{2.17}$$

En (2.17), δ_j corresponde al gradiente de la neurona j en una capa oculta, δ_k es aquel de las neuronas en la capa siguiente, w_{jk} es el peso entre cada una de las neuronas j y k y $f'\left(x_j\right)$ es la derivada de la función de activación. Cuando ya se dispone de los gradientes, los pesos se actualizan empleando el algoritmo de descenso de gradiente, que se expresa mediante (2.19).

$$\Delta w_{ij} = -\eta \cdot \delta_j \cdot d_i \tag{2.18}$$

donde η corresponde a un factor conocido como la tasa de aprendizaje, δ_j corresponde al delta (gradiente) de la neurona de salida y d_i es la entrada a la neurona.

Esta es la forma más usada para el ajuste de los pesos en redes neuronales, si bien, actualmente existen algunos algoritmos que no usan el descenso del gradiente debido

a ciertas limitaciones que puede llegar a tener, es también habitual que se empleen algoritmos basados en diferentes heurísticas. No obstante, este algoritmo es el empleado por excelencia y hasta la actualidad a permitido un gran desarrollo para el entrenamiento de redes neuronales.

2.4.3. Técnicas de Aprendizaje profundo (Deep Learning)

Con el incremento de la capacidad computacional que se ha obtenido en años recientes, el uso de modelos neuronales cada vez más complejos es posible. Actualmente, se pueden tener modelos neuronales con múltiples capas y un vasto número de neuronas en cada capa sin que esto sea un obstáculo para realizar tareas de entrenamiento que hasta hace pocos años, hubiesen sido irrealizables debido a su complejidad computacional.

A partir de esto, el Deep Learning ha sido usado ampliamente en algunas aplicaciones según [42], algunas de ellas son: visión por computadora, procesamiento de lenguaje natural, automatización y modelos predictivos. La variedad de aplicaciones también ha generado la invención de diferentes técnicas dentro del propio Deep Learning, esto se refiere a la existencia de distintas estructuras neuronales que ayudan a mejorar los resultados dependiendo la aplicación. Algunos de los tipos de redes neuronales empleadas en Deep Learning son: redes neuronales convolucionales, recurrentes, generativas adversarias, profundas y modulares. Las redes neuronales recurrentes a su vez poseen una división entre redes de memoria a corto y largo plazo.

2.4.3.1. Redes Neuronales Recurrentes

En [43], se indica que a diferencia de redes neuronales artificiales tradicionales, las cuales asumen la independencia entre los datos de entrada, las Recurrent Neural Network (RNN) realizan una captura de dependencias secuenciales y temporales. Un atributo definitorio de las RNN es la compartición de parámetros, esto permite al modelo llevar a cabo algunas inferencias acerca de secuencias de longitud variable. Sin dicha compartición, el modelo necesitaría parámetros únicos para cada dato en una secuencia. Las RNN agregan ciclos, los cuales conectan nodos adyacentes o pasos

de tiempo, generando una memoria interna la cual evalúa el dato actual a comparación de los datos previos. Esto ayuda a que las RNN lleven a cabo mapeos de uno a muchos, muchos a muchos y muchos a uno, en contraste con las redes neuronales tradicionales que se encuentran limitadas a mapeos de uno a uno entre la entrada y la salida. Las relaciones de recurrencia se ven generalizadas mediante (2.19), que indica que el estado del sistema es dependiente de un paso de tiempo anterior que esta dado por $t-1$.

$$S^{(t)} = f\left(s^{t-1}\right) \tag{2.19}$$

Esta ecuación puede ser reescrita como (2.20).

$$h^{(t)} = f\left(h^{t-1}, x^{t}\right) \tag{2.20}$$

Donde x^t corresponde a la entrada de una instancia de tiempo particular. La verdadera relevancia de h^t es que representa los aspectos relevantes de una secuencia pasada de entradas hasta t.

Las RNN tienen la capacidad de adaptarse a diferentes tipos de problemas en función de las secuencias de entrada y salidas necesitadas. Un buen ejemplo, corresponde a un escenario de análisis de series temporales en el mercado, donde la red es capaz de predecir el valor futuro de algunas acciones en base de los valores pasados o también prever los valores para cada día basándose únicamente en el valor del día anterior. Existen cuatro topologías que pueden adoptar las RNN: la primera es el Sequence-Vector, donde la red produce una única salida tomando en cuenta solo la información del último paso de tiempo; la segunda corresponde a la topología Sequence-Sequence, donde la red entrega resultados procesados en cada uno de los períodos de tiempo, esto agrega mayor complejidad en la salida; en tercer lugar se tiene a la topología Vector-Sequence, donde la red recibe un vector como entrada y genera una secuencia como salida; finalmente, la topología Encoder-Decoder, en la cual se combina dos topologías anteriores, pues el Encoder transforma una secuencia en un vector y el Decoder realiza el proceso contrario.

A pesar de que las RNN disponen de varias ventajas, poseen un problema conocido como “short-term memory”. Este problema se genera por el desvanecimiento del

gradiente durante el entrenamiento, esto produce que se generen dificultades para recordar información pasada mientras se procesa más elementos de la secuencia. Para poder abordar este problema, se ha propuesto el uso de celdas de memoria de corto y largo plazo Long Short-Term Memory (LSTM) [43], que son capaces de capturar patrones de secuencias más largas y se describen a continuación.

2.4.3.2. Redes Neuronales LSTM

Las redes LSTM corresponden a un tipo de redes neuronales recurrentes que están diseñadas para solucionar el problema del desvanecimiento de gradiente que experimentan las RNN tradicionales, [44]. Este tipo de redes emplean celdas de memoria que permiten mantener información en periodos de tiempo más largos, lo cual es crucial para tareas en las que se necesita recordar dependencias en un tiempo considerablemente amplio. Existen dos características principales de las redes LSTM: la primera corresponde a sus celdas de memoria, las cuales mantienen su estado en un período de tiempo más grande, esto permite almacenar información a largo plazo; mientras que la segunda característica corresponde a sus puertas de entrada, olvido y salida. Una puerta de entrada controla la información que ingresa a la celda de memoria, mientras que la puerta de olvido se encarga de decidir la cantidad de información anterior que tiene que olvidar y establecer la sección de información que debe entregarse como salida de la celda de memoria.

En una red LSTM la información es almacenada y procesada de dos maneras. La primera en una memoria de corto plazo, aquí la función de activación de una unidad neuronal es una función de la historia reciente del modelo, generando de esta manera una memoria a corto plazo. Por otro lado, los pesos de los enlaces entre neuronas también generan una memoria la cual se ajusta en base a la experiencia y el proceso de entrenamiento. Estos pesos se inician con un valor aleatorio y luego se optimizan para minimizar la función de pérdida. Esto crea la memoria a largo plazo.

Finalmente, las redes LSTM presentan dos beneficios importantes: *la resolución del problema del desvanecimiento del gradiente*, puesto que son capaces de recordar eventos que se han dado durante periodos largos, logrando así disminuir el impacto del desvanecimiento del gradiente; y, *la adaptabilidad* ya que son capaces de deter-

minar si el estado anterior de la red es relevante o si el estado actual sirve para otras unidades neuronales.

2.4.3.3. Redes Neuronales de Memoria a Corto Plazo (GRU)

Las redes neuronales recurrentes son las redes neuronales más adecuadas para capturar las relaciones entre datos secuenciales, sin embargo, es complicado mantener las dependencias a largo plazo utilizando RNN simples debido a que los gradientes tienden a desaparecer con secuencias largas, es aquí cuando se proponen los modelos de Gated Recurrent Unit (GRU).

En [45] se indica que las redes neuronales recurrentes GRU se requiere únicamente dos señales de *gating* (mecanismo que controla el flujo de información dentro de la red) con respecto al modelo LSTM . Las puertas son llamadas puerta de actualización z_t y puerta de reinicio r_t. El modelo RNN GRU se presenta de la siguiente forma:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.21}$$

$$\tilde{h}_t = g(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{2.22}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.23}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.24}$$

Desde (2.21) hasta (2.24) se puede observar el modelo GRU donde se presentan las compuertas de actualización y reinicio. También podemos notar que a pesar de que las redes GRU sean similares a las redes LSTM, estas tienen menos señales de *gating* lo que se traduce a una menor cantidad de parámetros necesarios. No obstante, las redes GRU tienen tres veces más parámetros en comparación a una RNN simple.

2.4.4. Redes Neuronales Profundas (DNN)

Estos modelos corresponden a un tipo de red neuronal artificial, la cual contiene un conjunto de capas ocultas entre las capas de entrada y salida, [46]. Al igual que las

ANN poco profundas, las Deep Neural Network (DNN) son capaces de modelar relaciones no lineales muy complejas.

Para el entrenamiento de este tipo de redes se emplea comúnmente aprendizaje supervisado o aprendizaje por refuerzo. La optimización de los modelos generalmente se lleva a cabo mediante optimizadores que emplean el método de descenso de gradiente para lograr optimizar la red y minimizar la función de pérdida. Aquí también entra en juego el algoritmo de retropropagación, como el más popular para el entrenamiento en modelos de Deep Learning.

Existen varias aplicaciones que pueden hacer uso de este tipo de redes, entre ellas se puede mencionar la clasificación de imágenes, el reconocimiento de objetos, reconocimiento facial, diagnósticos médicos entre otras.

2.5. Redes Neuronales para la codificación de canal

Está Sección se centra en presentar algunos trabajos e investigaciones que se han llevado a cabo sobre el uso de redes neuronales para la codificación de canal, en primer lugar se realiza una investigación general acerca del uso del Deep Learning en diferentes aplicaciones en el área de las comunicaciones digitales. Después se centra en el uso de estas técnicas específicamente en el campo de estudio de la codificación de canal, se observan algunas arquitecturas empleadas para el entrenamiento de los modelos, luego se presenta un enfoque específico de Deep Learning para decodificadores y finalmente se menciona el uso de Python como herramienta fundamental para el desarrollo de escenarios de entrenamiento y prueba para redes neuronales.

2.5.1. Deep Learning como factor de progreso en las comunicaciones digitales

En [47], presentan algunas aplicaciones innovadoras basadas en Deep Learning centradas en la capa física. Aquí adoptan una nueva visión para los sistemas de comunicaciones, esta visión establece entender al sistema como un autoencoder, en base a ese planteamiento, se genera un nuevo método para idear el diseño de sistemas de comunicaciones, donde se propone entender el diseño como una tarea de

reconstrucción entre el transmisor y el receptor y el objetivo es la optimización de los componentes en un mismo proceso.

Esta idea puede ser exportada a arquitecturas de múltiples transmisores y receptores, lo que lleva al concepto de redes de transformadores de radio como un método de inserción de conocimiento experto del dominio en el modelo de Deep Learning. También se presenta la aplicación de Convolutional Neural Network (CNN) en muestras “In-Phase” y “Quadrature” (IQ) crudas para llevar a cabo la clasificación de modulación, esto genera una precisión comparable a esquemas tradicionales que son dependientes de características expertas.

Lo más relevante en [47], sin duda es la introducción de una nueva idealización de las comunicaciones, donde se pretende traducir el proceso de diseño a la optimización de una tarea de reconstrucción de extremo a extremo, con el objetivo de aprender las implementaciones del transmisor, receptor y también de las codificaciones de señales que realizan los modelos sin ningún tipo de conocimiento previo.

Comparando este tipo de aplicaciones, con métodos convencionales, se puede ver en varios escenarios que el rendimiento del BLER es competitivo, sin embargo, la escalabilidad a longitudes de bloque largas representa un desafío.

Así mismo, lo novedoso de este enfoque es que puede entregar información de interés acerca de esquemas de comunicación óptimos, en escenarios completamente desconocidos. Esto marca un claro comienzo para una variedad de estudios en el ámbito del Deep Learning y el Machine Learning que pueden converger en nuevos esquemas de comunicaciones que ofrezcan mejores características que los empleados en la actualidad.

En [48], realizan un recuento de diferentes aplicaciones de las redes neuronales en el área de comunicaciones digitales, tal como identificación y ecualización de canales, codificación y decodificación, cuantificación vectorial, entre otros. Mencionan que el elemento fundamental en modelos que emplean redes neuronales es hallar una arquitectura correcta capaz de brindar los mejores resultados. Se presenta como llevar a cabo la elección de estructuras de redes neuronales y el proceso para combinar los algoritmos generados con técnicas como procesamiento adaptativo de señales, siste-

mas difusos y también algoritmos genéticos. Además se revisan los enfoques matemáticos que se usan para tener una compresión estructurada del comportamiento de aprendizaje y la convergencia de los algoritmos generados por las redes neuronales.

El punto con mayor grado de importancia para los intereses del trabajo que estamos realizando es la codificación, decodificación y detección de errores. En [48] se comenta que estos son campos sumamente prometedores para las aplicaciones de redes neuronales, en consecuencia a la necesidad de realizar cálculos paralelos, se presentan los requerimientos de grandes velocidades de cómputo. Los autores de [48] comentan varias aplicaciones halladas para las redes neuronales en el contexto de la corrección de errores. Una de las aplicaciones comentadas es una red neuronal generada para la decodificación de canales con ruido blanco gaussiano aditivo para el código Bose–Chaudhuri–Hocquenghem (Códigos) (BCH) (7,4). Por otro lado, también se presenta un decodificador de Viterbi basado en ANN. Esta aplicación es de gran interés puesto que este no solamente iguala el rendimiento de un decodificador de Viterbi ideal sino que además debido a la naturaleza de las neuronas implementadas en el decodificador, permite que este tenga un comportamiento mucho más rápido.

En [49], se centran en la problemática de aprender algún tipo de codificación para la capa física que sea eficiente y adaptable, con el objetivo de transmitir información por medio de un canal deteriorado. Aquí el problema se aborda desde punto de vista del aprendizaje automático no supervisado, en vez de metodología tradicional. La tarea central es optimizar la pérdida de reconstrucción a través de capas de deterioro artificial en el autoencoder, así mismo, se agregan otras capas de regularización, estas se encargan de emular degradaciones frecuentes en los canales inalámbricos. También se habla sobre cuál es la función de los modelos de atención en la forma de la red transformadora de radio para la recuperación de representaciones canónicas de señales previo al proceso de decodificación.

La importancia que representa aprender a transmitir información de forma óptima en base al canal de comunicación sin esquemas de modulación altamente complejos es gigantezca. A pesar que teóricamente se ha comprobado la existencia de códigos que sean capaces de alcanzar la capacidad del canal y tras millones de dolares de inversión aún no se ha encontrado una solución general para todos los escenarios y en

algunos casos la dificultad que representa implementar algunos algoritmos diseñados para acercarse a la capacidad deseada es muy alta. A pesar de ello, en [49], se muestra que la arquitectura basada en redes neuronales es viable para ser implementada y posee un potencial de rendimiento capaz de competir con sistemas de comunicación actuales, casi alcanzando la capacidad de Shannon, pero manteniendo generalidad y baja complejidad.

Así mismo, se presenta algunos descubrimientos experimentales, como el impacto que representa la elección de SNR de entrenamiento y la tasa de Dropout, esta parametrización de modelos también es un área de estudio compleja y digna de investigación para encontrar metodologías de entrenamiento eficientes para modelos de redes neuronales.

En general, se puede ver que el uso de este tipo de técnicas tiene una pendiente positiva para el descubrimiento y evolución de los sistemas de comunicación actuales y sus componentes.

En [50], los autores proponen un nuevo algoritmo para decodificar códigos lineales con el uso de Deep Learning. Este método mejora el algoritmo de propagación de creencias estándar asignando pesos a los bordes del gráfico de Tanner, donde los bordes son generados con el uso de aprendizaje profundo.

Este método propuesto presenta resultados positivos puesto que se logra una mejora en el BER al ponderar de manera adecuada los mensajes transmitidos. Cabe mencionar que el decodificador propuesto una vez entrenado, puede mejorar el desempeño en comparación con el algoritmo de propagación de creencias estándar sin la necesidad de aumentar la capacidad computacional. Los autores consideran este trabajo como uno de los primeros pasos en la implementación del Deep Learning para el diseño de decodificadores mejores.

En [51] podemos observar una investigación similar en la que se estudian distintas técnicas para la decodificación de códigos lineales. En este trabajo se ha demostrado que el Deep Learning puede ser usado para mejorar un decodificador de propagación de creencias estándar. Además se ha introducido una arquitectura de decodificador neuronal recurrente basada en el método de relajación sucesiva, en este decodifica-

dor se observan mejoras sobre el algoritmo de propagación de creencias estándar. En [51] se obtuvieron mejoras sobre los decodificadores estándar de propagación de creencias con el uso de redes neuronales y se introducen decodificadores que mantiene el equilibrio entre rendimiento a la hora de corregir errores y la complejidad de la implementación.

2.5.2. Generación de códigos mediante el uso de aprendizaje profundo

En [7], se propone el uso de técnicas de Deep Learning para la codificación sobre un canal AWGN, uno de los escenarios más prácticos ya que estándares como 5G Long Term Evolution (LTE) lo toman como referencia para la creación de los mismos. La propuesta comprende un codificador de canal que recibe K bits de información y luego los convierte en n señales reales para poder transmitirlas por medio de un canal ruidoso. Para cada uso del canal (denotado por i), cada uno de los símbolos x_i trasmitidos es afectado por ruido aditivo Gaussiano independiente e idénticamente distribuido para cada uso del canal, y denotado por n_i. Para este modelo de canal, el decodificador recibe $y_i = x_i + n_i$. Luego de recibir los n símbolos, el decodificador determina cuál de las secuencias de bits fue enviada, entre las 2^K opciones posibles, con el objetivo de maximizar la probabilidad de una correcta decodificación.

En canales caracterizados por ruido blanco Gaussiano (AWGN), tanto el codificador como el decodificador corresponden a funciones que tienen como objetivo mapear bits de información a señales reales, mientras que las señales recibidas se mapean a bits de información respectivamente. En [7], indican de forma general que el principal reto que se tiene al momento de diseñar un buen código es lograr una baja tasa de error y que sea eficiente en términos de su complejidad computacional.

Tomando como referencia los trabajos presentados en [7] y [8], donde el enfoque central es el análisis del BER, en este trabajo de Integración Curricular se busca de igual manera evaluar el desempeño de modelos neuronales de codificadores y decodificadores en términos de este parámetro. Si bien un estudio comprensivo de la complejidad computacional está fuera de los objetivos primordiales de esta investigación, se realiza la medición de los tiempos de ejecución sobre un mismo equipo para diferentes códigos contemplados en este trabajo con el fin de observar la carga

computacional. La Sección 4.3, presenta una evaluación experimental de los tiempos de procesamiento en la función de codificación/decodificación de lotes de información para los modelos de redes neuronales que presentaron resultados comparables con códigos convencionales y las implementaciones de dichos códigos. Esta evaluación permite contrastar la duración de codificación y decodificación, de los modelos neuronales y tradicionales, ejecutándose en un mismo equipo computacional (plataforma Colab), y con ello, brinda una perspectiva del esfuerzo computacional de cada uno de estos enfoques.

Si bien décadas de progreso en el campo de la teoría de codificación han resultado en el desarrollo de códigos innovadores tales como los convolucionales, Turbo, LDPC y polares, estos solo se acercan a los límites fundamentales de una comunicación confiable.

En particular [7] se enfoca en un canal AWGN canónico con retroalimentación ruidosa (inclusión de un canal de feedback), donde el símbolo que se recibe y_i, luego se transmite de regreso al codificador con un retraso de una unidad de tiempo por medio de otro canal de ruido Gaussiano aditivo (aunque el escenario de retroalimentación ideal también puede ser considerado). Esta revisión será la referencia del planteamiento de este trabajo de titulación, donde se explora el uso de un BSC. Dada la disponibilidad de un canal de retroalimentación, el codificador puede emplear las transmisiones recibidas mediante *feedback* para decidir de forma adaptativa que símbolo debe transmitir a continuación.

En [7] se sugiere que una estrategia natural para la creación de un código con retroalimentación es emplear una red neuronal recurrente RNN como codificador, esto porque la comunicación con retroalimentación es un proceso secuencial. De esta manera tanto el codificador y el decodificador se presentan como dos modelos basados en RNN's, siguiendo el principio fundamental de un auto-encoder, donde se entrenan de manera simultánea tomando en cuenta el intercambio de información en las dos direcciones mediante canales AWGN, donde la retroalimentación es ruidosa. El objetivo del entrenamiento es minimizar el error en la decodificación de los bits de información. Este enfoque requiere atravesar un proceso experimental para seleccionar de manera cuidadosa varios de los elementos de diseño al construir, parametrizar y entrenar un

modelo basado en RNN y que permite descubrir nuevos códigos.

Los autores contemplan un escenario para una transmisión a un *rate* de 1/3, en el que se usa palabras codificadas en binario de longitud 50 bits, que corresponden a los mensajes, y 100 bits adicionales que se generan como redundancia mediante el codificador.

A partir de un modelo base, los autores buscan diferentes maneras de obtener una mejora de rendimiento teniendo en consideración un esquema de codificación para canales con retroalimentación. Esto induce a un planteamiento de dos fases en la etapa de codificación, en la primera fase solo se envía un bloque de símbolos correspondientes a la misma palabra a ser transmitida, para después, transmitir un código basado en la retroalimentación, el mismo que es generado mediante un modelo de RNN. De esta manera, el código solo necesita diseñarse para la segunda fase. En [7], se lleva a cabo un proceso, donde se plantea un esquema base y el cual va obteniendo mejoras progresivas. En el primer esquema experimental propuesto, no se tiene más que el encoder y decoder diseñados con una capa RNN y una capa lineal de salida con su respectiva función de activación. En un segundo esquema propuesto, se realiza una mejora que denominaron *zero padding*, en la que se envían bits adicionales en respuesta a la observación de resultados donde los últimos bits tenían un menor rendimiento en cuanto al BER (puesto que no contaban con feedback secuencial), por ello, al agregar unos bits irrelevantes al final de cada palabra suponían que podrían obtener una mejora en ese sentido. Un tercer esquema propone el uso de una capa adicional de pesos promedios la cual mejora de manera general el BER y finalmente, agregan una capa adicional la cual realiza un balance del BER entre los primeros y últimos bits.

Las mejoras de rendimiento se van construyendo sobre el esquema base de codificador y decodificador, consiguiendo así una red neuronal balanceada que genera buenos resultados. En la Figura 2.14, se observa el resultado del BER en función del SNR de cada una de las pruebas realizadas de la red neuronal y también la comparación con algunos códigos convencionales para el mismo modelo de canal objeto de dicho estudio.

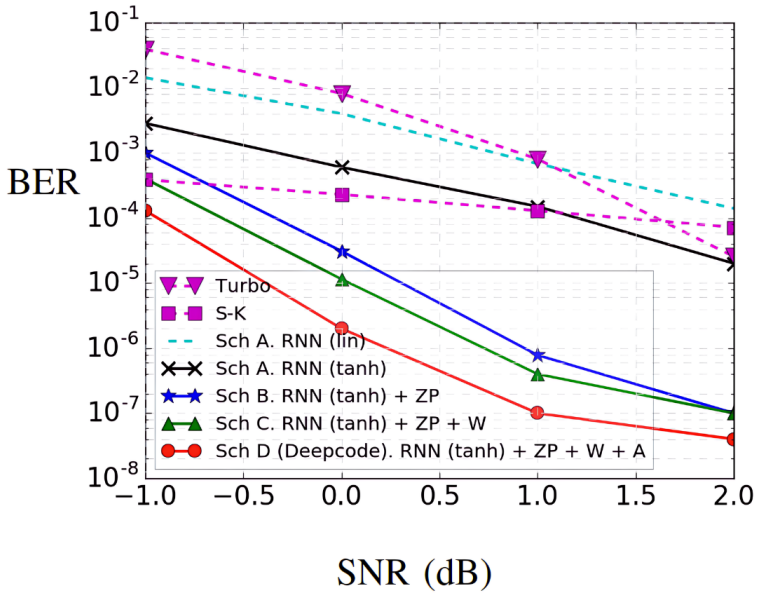

Figura 2.14: BER en función del SNR [7], © 2020 IEEE.

En La Figura 2.14, se observa que las mejoras en la arquitectura de los modelos neuronales consiguen una reducción mayor del BER, tal como se mencionó anteriormente. De igual manera, en La Figura 2.15, se observa el BER por bit transmitido y también se puede presenciar la evidente mejora con los distintos ajustes en el esquema base.

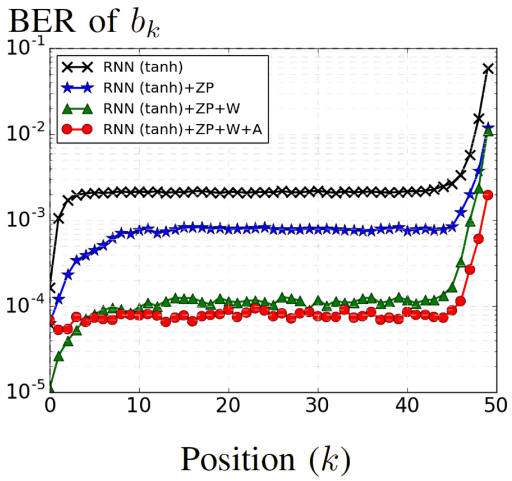

Figura 2.15: BER por bit [7], © 2020 IEEE.

De manera interesante, la metodología de prueba y análisis que adoptan para poder ir dando diferentes estructuras a la red, resulta en mejoras progresivas de los resultados que se esperan.

Por último, en [7], llevan a cabo una interpretación de los resultados obtenidos, con el fin de buscar una explicación sobre el código descubierto por el modelo basado en RNN. Los autores atribuyen la segunda fase, en la que el codificador está enfocado en mejorar los bits de información dañados por el ruido en la primera fase y un comportamiento particular, en la que los bits componentes del código responden a un patrón que depende de la información pasada y presente. El esquema de codificación de dos fases propuesto tiene como objetivo corregir los problemas causados por el ruido que se introduce en la primera fase. Para la segunda fase el codificador puede aprender a transmitir información buscando ayudar al decodificador en la correcta estimación de los bits que se dañaron por efectos del ruido.

La Figura 2.16, muestra que los bits de redundancia creados por el codificador responden a un comportamiento esencialmente determinista que se asimila a una función ReLu. Los bits $c_{k,1}$ y $c_{k,2}$ se generan en función del ruido percibido y del bit c_k transmitido en la primera fase del código.

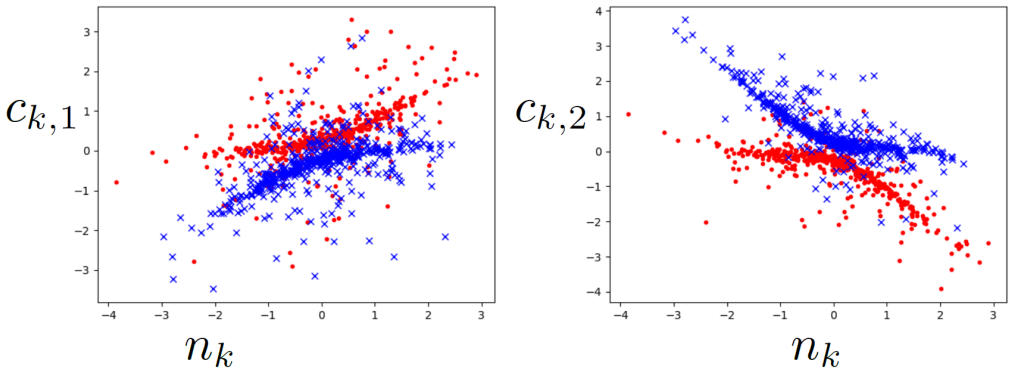

Figura 2.16: Reacción de bits de paridad generados por la red [7], © 2020 IEEE.

Estos resultados también evidencian que el codificador hace uso de la memoria del modelo RNN, sugiriendo que el código explota las características esenciales de un buen código de retro-alimentación. Con base en este trabajo, se fundamenta los puntos clave del desarrollo de este trabajo de titulación, pues la idea es evaluar de la misma manera cómo esta metodología tiene efectos positivos sobre un modelo diferente de canal, en particular sobre BSC.

2.5.3. Otras arquitecturas basadas en Deep Learning y feedback

En [8], se continua una línea de investigación similar a la propuesta en [7], pero con una nueva arquitectura de codificación de errores. La propuesta está basada en el uso de las DNN para canales con retroalimentación, que se denomina Deep Extended Feedback (DEF). Este tipo de codificador envía la secuencia correspondiente a la secuencia original seguida de una secuencia de símbolos redundantes los cuales están generados en función del mensaje y de las observaciones de las salidas del canal de *feedback* que son enviadas hacia el transmisor. Estos códigos DEF generalizan Deep Code, pues los símbolos de paridad son producidos por observaciones del canal durante intervalos de tiempo mucho más largos para lograr mejorar la capacidad de corrección de errores y también se emplean formatos de modulación los cuales aumentan la eficiencia espectral.

En sistemas de codificación convencionales, la corrección de errores se logra con el uso de códigos binarios lineales, combinados con mecanismos de retransmisión tal como Hybrid automatic repeat request (HARQ). Lamentablemente, el uso de mensajes simples de ACK/NACK en HARQ no aprovecha completamente el canal de retroalimentación, puesto que no hace un uso eficiente del mismo. Generalmente los códigos que hacen un uso completo de la retroalimentación logran un rendimiento mejorado en comparación con códigos convencionales, sin embargo, siempre debe considerarse la calidad del canal de retroalimentación, el mismo que en la práctica, no puede ser perfecto (noiseless).

La obtención de buenos códigos para canales con retroalimentación generalmente es un problema notoriamente difícil. Varios métodos de codificación para canales con retroalimentación han sido propuestos, sin embargo, todas las soluciones conocidas no logran el rendimiento que se prevé o exhiben una complejidad inasequible. Recientemente, se ha obtenido progreso significativo aplicando métodos de aprendizaje automático, donde el codificador y decodificador se implementan como dos DNN separadas. Los coeficientes de las DNN son calculados mediante un proceso de entrenamiento de codificador que se realiza mutuamente. El nuevo código que está basado en DNN para canales con retroalimentación denominado DEF y documentado en [8] está basado en la extensión de la retroalimentación, esto consiste en extender la en-

trada del codificador para incluir versiones retrasadas de las señales de retroalimentación. De esta manera, el codificador DEF comprende la señal de retroalimentación más reciente y un conjunto de señales de retroalimentación pasadas dentro de una ventana de tiempo.

El código DEF combina diferentes arquitecturas de redes neuronales recurrentes, tales como RNN, GRU y LSTM . Los principales beneficios del código DEF incluyen una mejor capacidad de corrección de errores que se obtiene a través de la extensión de la retroalimentación y una mayor eficiencia espectral gracias al uso de modulaciones Quadrature Amplitude Modulation (QAM) /Pulse Amplitude Modulation (PAM). Así mismo, se evalúa canales que tienen retroalimentación con ruido y sin ruido. En la primera fase de este código DEF el mensaje se convierte en una secuencia de símbolos reales y se transmite. El receptor los recibe con ruido y los devuelve al transmisor colocando más ruido (ya que el canal de feedback no es perfecto). El codificador DEF emplea un modulador QAM /PAM y un Parity Symbol Generator (PSG) que usa redes neuronales como RNN, GRU o LSTM para poder realizar el cálculo de símbolos de paridad partiendo de observaciones del canal en períodos de tiempo más largos. El decodificador por su parte DEF emplea una red neuronal recurrente bidireccional para llevar a cabo el mapeo de los símbolos recibidos al original, logrando mejorar la protección contra errores distribuidos de manera desigual.

Naturalmente, las dos redes DNN (codificador + decodificador), son entrenadas en conjunto. Los bloques de mensajes que se transmiten se generan aleatoriamente. El optimizador Adaptive Moment Estimation (ADAM) actualiza los coeficientes de las DNN empleando la función Binary Cross Entropy (BCE). La tasa de aprendizaje al principio es 0.02 y disminuye a la décima parte luego de 10^3 bloques de mensajes del conjunto de entrenamiento.

Algunos de los resultados obtenidos por el código planteado en [8] se muestran en la Figura 2.17, donde se ilustra que el código DEF con entrada del codificador extendida tiene un mejor rendimiento que Deep Code. Los códigos DEF-LSTM tienen el mejor rendimiento entre los códigos evaluados.

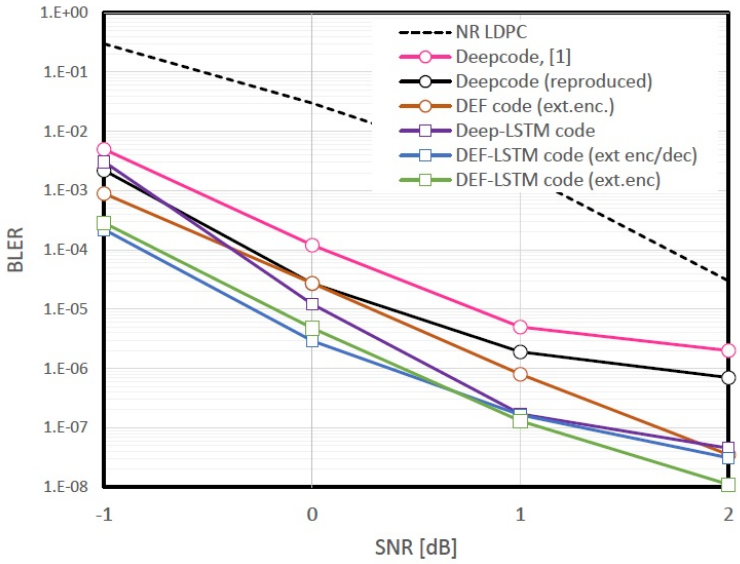

Figura 2.17: Resultados de rendimiento entre Deep Code, códigos DEF, Deep Code basado en LSTM y códigos DEF-LSTM [8], © 2021 IEEE.

En relación a la estabilización del entrenamiento, habitualmente se emplea un conjunto grande de datos para estabilizar y acelerar la convergencia, así mismo, un mecanismo de retroceso es el que descarta las actualizaciones de pesos si la pérdida es mayor que la obtenida en la anterior iteración.

En [8] se menciona con base a observaciones experimentales que una posibilidad de mejorar el rendimiento de un modelo neuronal es realizar el entrenamiento con mensajes de longitud mayor a los mensajes que se utilizarán durante la evaluación del modelo. Específicamente, utilizar durante el entrenamiento mensajes con el doble de longitud de los mensajes empleados en la evaluación, resulta beneficioso, por ejemplo, con mensajes del lote de prueba de 50 bits de longitud y una longitud para mensaje en el entrenamiento de 100 bits. No obstante, este beneficio se desvanece cuando el entrenamiento involucra lotes de mensajes sustancialmente más grandes, que poseen más palabras, por lo que finalmente se sugiere que tanto los mensajes del lote de entrenamiento, como los del lote de prueba tengan la misma longitud y se opere con lotes de palabras extensos (por ejemplo, en el orden de las 20000 palabras). Este hecho se ratifica en [7] (Sección 4), donde también se estudia los efectos de la longitud de los mensajes durante las fases de entrenamiento y prueba. La motivación de estudiar estos efectos se fundamenta en los estándares de comunicaciones inalámbricas actuales donde existe una cantidad importante de longitudes de bloque

que son de interés práctico; por ejemplo, en el estándar LTE los bloques pueden tener una longitud entre 40 y 6144 bits. Observe la Figura 5 (centro) en [7], donde se presenta el desempeño de un sistema de codificación basado en redes neuronales, empleando mensajes con una longitud de palabra de 50 y 500 bits, en un escenario con retroalimentación sin ruido para canales AWGN. El BER no experimenta mejoras considerables ante un incremento de la longitud de los mensajes a codificar. Sin embargo, esta observación no puede generalizarse de manera absoluta puesto que sí es posible diseñar códigos que mejoran BER a medida que la longitud de bloque aumenta. Un ejemplo de este hecho son los Turbo Códigos, los cuales poseen una tasa de error que disminuye exponencialmente con la longitud de bloque, tal como se menciona en [7].

2.5.4. Un enfoque Deep Learning para el decodificador

En [52] se propone un escenario distinto a los descritos en las anteriores subsecciones, pues aquí se hace uso de redes neuronales solo en el decodificador, mientras que en la etapa de codificación se puede usar cualquier método convencional, como codificación con turbo códigos, códigos convolucionales o LDPC . Un enfoque en la misma línea de trabajo se desarrolla en [51] donde se usan métodos de aprendizaje profundo para mejorar la decodificación de códigos lineales .

En [52] se plantea usar una Doubly Residual Neural (DRN) para la decodificación de canales. Básicamente la arquitectura de la DRN consta de dos partes, una CNN y una Fully Connected Network (FCN). La CNN es empleada para procesar la entrada de la red, que es un vector de mensajes de canal y la FCN en cambio se usa para generar la salida de la red, y corresponde a un vector de bits decodificados. La idea es poder integrar tanto la entrada residual como el aprendizaje residual en la arquitectura de la red neuronal y de esta forma poder mejorar el rendimiento de la decodificación.

Detallando un poco más la arquitectura de la DRN se puede decir que esta emplea la entrada residual y el aprendizaje residual en ambas partes de la red. En la CNN, la entrada residual se junta con la entrada original antes de la primera capa convolucional, en cambio, en la FCN, la entrada residual se agrega a la entrada de la primera capa completamente conectada. Así mismo, el aprendizaje residual se usa en todas

las capas de la red, esto permite que la red se centre solo en aprender las características relevantes de los datos. Para el entrenamiento de la DRN se emplea un algoritmo de descenso de gradiente estocástico o Stochastic Gradient Descent (SGD) con retropropagación.

Mientras se ejecuta el entrenamiento, la función de pérdida de la red es minimizada. Esta función es la que mide la diferencia entre la salida deseada y la salida actual de la red. Los resultados experimentales dejan ver que el DRN supera a los decodificadores neuronales actuales en términos de rendimiento de decodificación, tamaño de modelo y costo computacional. Particularmente, el uso de DRN en el decodificador logra un BER más bajo que otros decodificadores neuronales en varios tipos de códigos de canal, entre ellos incluidos los códigos LDPC, Polar y BCH. Esto resulta interesante para nuestra propuesta debido a que muestra una arquitectura diferente e igualmente óptima para la transmisión de datos.

En [53] se presenta una visión general del uso de distintas técnicas de Deep Learning para la codificación de canales, este documento es muy interesante porque logra diferenciar bajo qué condiciones las diferentes técnicas pueden funcionar eficientemente, mientras que en [54] se habla sobre la arquitectura interna de las redes neuronales. Específicamente se indica que las DNN son aplicables para la decodificación de secuencias binarias en la codificación de canales. Pues como se sabe, a través de un canal viajan unos y ceros, que debido a interferencias o ruido pueden cambiarse y la idea es que el decodificador DNN sea capaz de decodificar estos datos, en vez de usar decodificadores tradicionales. Para ello, el decodificador con DNN tiene una etapa de entrenamiento donde va ajustando sus pesos y *bias* para lograr minimizar el error de predicción entre la secuencia original y la secuencia decodificada.

Por otro lado, las CNN se pueden usar para la decodificación de secuencias de símbolos. Esto se debe a que las CNN son capaces de aprender patrones complejos en los datos de entrada y de generalizar a datos nuevos. Así mismo, pueden manejar grandes cantidades de datos y procesarlos en paralelo, esto los hace adecuados para aplicaciones en tiempo real.

En términos generales, el uso de estas técnicas en la codificación de canal pueden mejorar en gran medida el rendimiento de decodificación, reducir la complejidad

computacional, el consumo de energía y mejorar la latencia de procesamiento. Las CNN pueden ser empleadas especialmente para tratar con la distorsión no lineal del canal, puesto que son capaces de extraer características relevantes de los datos de entrada y poder compensar esas distorsiones. El uso de RNN para procesar datos secuenciales se ha vuelto atractivo en años recientes debido a que tienen la capacidad de recordar información de estados anteriores y emplearlo para procesar datos futuros, esto hace que sean útiles para tareas de procesamiento de datos secuenciales. Las RNN son capaces de aprender a modelar la estructura recurrente de turbo códigos o códigos convolucionales y esto permite mejorar la precisión en la decodificación.

Finalmente, en la actualidad el uso de técnicas de Deep Learning ha aumentado debido al gran poder computacional que se ha desarrollado y son métodos que van tomando fuerza para ser aplicados en la transmisión de información. En ese contexto, se puede encontrar investigaciones relacionadas con diferentes arquitecturas que contribuyen a la propuesta de este trabajo de titulación y que marcan un camino de investigación con pasos bien definidos para alcanzar diferentes objetivos.

2.5.5. Python como herramienta para la implementación de redes neuronales

Desde su creación, el lenguaje Python ha demostrado ser una herramienta muy potente para el desarrollo de la ciencia de datos. Esto se debe principalmente a la incursión de la comunidad en el desarrollo de paquetes que faciliten la implementación de redes neuronales. Según [55], el lenguaje de programación más empleado en el ámbito de la IA es Python. Además de ser una herramienta de código abierto u open-source, Python ofrece todas las funcionalidades para el desarrollo de este trabajo.

En [56], se indica que actualmente coexisten varias librerías para la implementación de modelos de aprendizaje automático. A pesar de ello, hay tres que destacan estas corresponden a: Keras, TensorFlow y PyTorch.

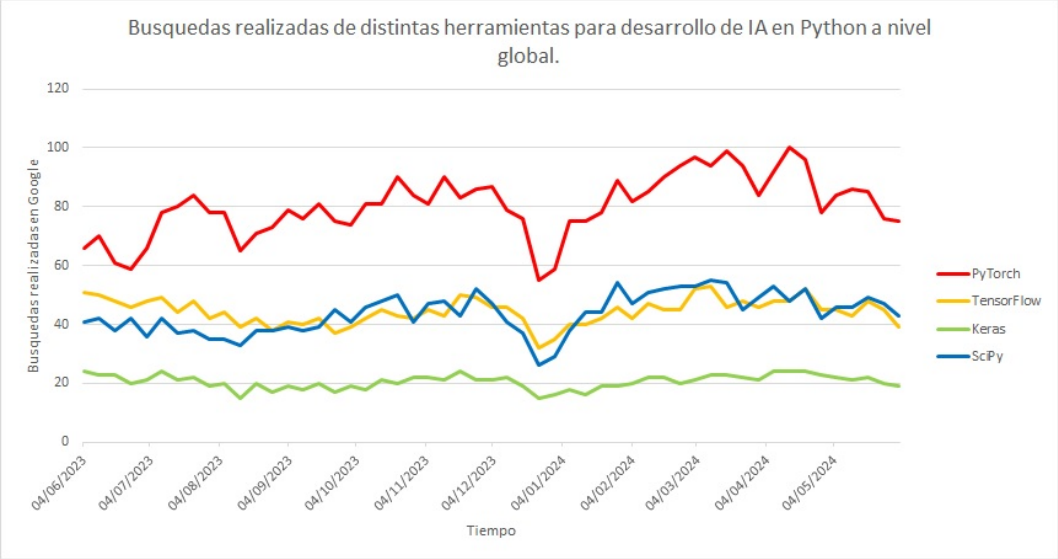

Figura 2.18: Tendencias en el uso de librerías de IA en Python del 04/06/2024, generado con Google Trends [9].

En función de la Figura 2.18, se puede notar que Pytorch[1] es actualmente la librería más usada y analizada a nivel mundial, lo que se empareja con las revisiones que los autores de este trabajo realizaron para seleccionar y estudiar distintas librerías, concluyendo que existe una comunidad más desarrollada y madura entorno a PyTorch. Es así que este conjunto de librerías fue seleccionado como la herramienta base para el desarrollo de nuestro trabajo de titulación. A continuación se revisa brevemente el funcionamiento y uso general de este Framework para el desarrollo de redes neuronales, además de destacar sus cualidades principales.

PyTorch corresponde a una biblioteca diseñada para desarrollos en lenguaje Python, la cual permite la construcción de proyectos relacionados con el aprendizaje profundo [11]. Se caracteriza por tener gran flexibilidad y permitir expresar modelos de aprendizaje profundo en un estilo semejante a lo esperado en cuanto sintaxis de Python. De hecho, la accesibilidad y facilidad de uso es lo que ha llamado la atención de la comunidad científica, convirtiéndose así en una de las herramientas más empleadas y preferidas dentro de la gama de aplicaciones de aprendizaje profundo.

Este framework brinda una excelente introducción para el aprendizaje profundo, esto hace que sea adecuado para contextos educativos y profesionales. La sintaxis clara,

[1] En [57], indican que Pytorch es una librería de código abierto, que tiene como idioma base Python y cuyo objetivo es ser empleada en proyectos de aprendizaje automático. Está especializada en cálculos de tensor, aceleración de GPU y diferenciación automática.

API simplificada y la facilidad de depuración convierten a PyTorch en una excelente opción para principiantes. Adicionalmente, el núcleo de información de PyTorch está compuesto por elementos conocidos como tensores, que corresponden a arreglos multidimensionales similares a los arrays de NumPy. Los tensores se diferencian de otros tipos de elementos porque con ellos se pueden realizar operaciones matemáticas aceleradas en hardware especializado, esto permite que el proceso de diseño y entrenamiento de las redes neuronales sea sencillo tanto en máquinas individuales como en recursos de computación paralela.

En el contexto del aprendizaje profundo, Pytorch permite abordar tareas complejas, comprendiendo operaciones de traducción automática, identificación de objetos en escenas desordenadas, etc. Para la implementación de estas aplicaciones inteligentes, se necesitan herramientas flexibles, eficientes y capaces de abordar la variabilidad en datos de entrada. Por ello, algunas de las razones por las que PyTorch es una herramienta que cumple con esas características son las siguientes:

- Simplicidad: PyTorch es sencillo de aprender, usar, extender y depurar, esto lo transforma en accesible para investigadores y profesionales.
- Naturalidad en la programación: PyTorch usa como su átomo de datos el Tensor para el manejo de números, vectores, matrices o arreglos en general, junto con funciones las cuales permiten operar sobre ellos. Además permite la programación incremental e interactiva, muy similar a Numpy.
- Cálculo acelerado: Esto se refiere a una opción integrada en la librería la cual permite la aceleración mediante el uso de Graphics Processing Unit (GPU), consiguiendo de esta forma aumentos significativos en la velocidad de ejecución y procesamiento a comparación con cálculos en Central Processing Unit (CPU).
- Optimización numérica: Brinda facilidades para la optimización numérica para expresiones matemáticas genéricas y cruciales para el entrenamiento en aprendizaje profundo.
- Alta performance: PyTorch es un framework de alto rendimiento con soporte de optimización para la computación científica.
- Expresividad: Permite implementar modelos complejos sin la necesidad de im-

poner una complejidad indebida, lo que se traduce en una interpretación de ideas sencilla en código de Python.

- Transición a producción: A pesar que en sus inicios PyTorch estaba centrado en la investigación, en la actualidad incluye un entorno de ejecución de alto rendimiento en C++ para lograr desplegar modelos sin depender directamente de Python, además admite el diseño y entrenamiento de modelos en C++.

Por otro lado, [10] provee una visión más sencilla acerca de PyTorch, pues plantea a este Framework, como el resultado de la suma de tres componentes principales, tal como se muestra en (2.25).

$$PyTorch = Numpy + Autograd + GPU \tag{2.25}$$

PyTorch sigue una interfaz similar a la de Numpy, y recordando que Numpy es la librería por excelencia para la ciencia de datos en Python, se puede establecer que, si se conoce el uso de Numpy, entonces, la curva de aprendizaje de PyTorch tendrá una pendiente no tan elevada y no debería implicar gran complejidad. De esta forma, la mayor parte de los conceptos que se manejan sobre Numpy pueden aplicarse sobre PyTorch, esto incluye las operaciones matemáticas, indexado y troceado, iteración, vectorización y broadcasting.

Si bien, la facilidad de uso y la funcionalidad de una estructura de datos eficiente que brinda PyTorch es muy relevante, la característica más importante de este Framework es el Autograd. El Autograd ofrece la funcionalidad de calcular derivadas de forma automática con respecto a cualquier tensor. Esto brinda un gran potencial para el diseño de redes neuronales complejas y también para el entrenamiento empleando algoritmos de gradientes sin la necesidad de calcular todas las derivadas de forma manual. Para realizar estas operaciones, PyTorch construye de forma dinámica un grafo computacional. Es decir, cada vez que se efectúa una operación sobre uno o varios tensores, éstos se agregan al grafo computacional junto a la operación en concreto. De esta forma, si se desea calcular la derivada de cualquier valor con respecto a cualquier tensor, lo único que se debe realizar es aplicar el algoritmo de retropropagación en el grafo. El algoritmo de retropropagación es la regla de la cadena de la

derivada.

La mejor manera de entender acerca de la generación del grafo computacional y el algoritmo de retropropagación, es mediante un ejemplo, para ello, se explica el desarrollo de [10]. Tomando como ejemplo la Figura 2.19, se observa algunas operaciones que se realizan con los tensores principales x, z e y.

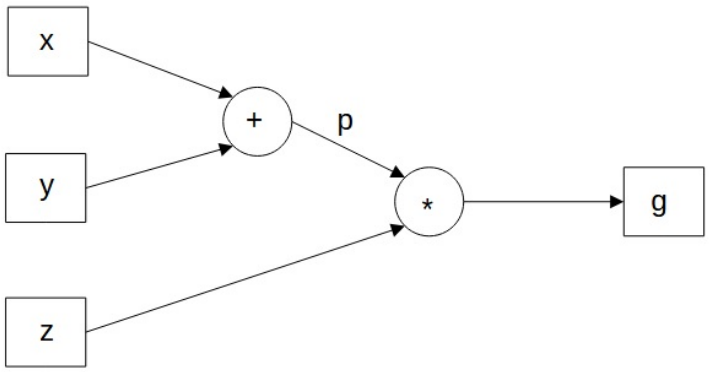

Figura 2.19: Grafo del ejemplo descrito, imagen adaptada de [10].

Arriba puede observarse con facilidad que p, es el resultado de $x + y$ y que luego el tensor intermedio p es empleado para obtener el resultado final $g = p \times z$. Entonces, cada vez que se aplica una operación sobre un tensor que tiene activada la opción de Autograd, PyTorch irá construyendo y actualizando el grafo computacional. Lo que se presenta en la Figura 2.19, es como se observaría el grafo computacional en este ejemplo específico. Luego, simplemente se debe correr el algoritmo de retropropagación implementado en PyTorch y con esto ya se podría acceder a las diferentes derivadas del grafo. El grafo computacional sin duda es una herramienta estupenda para el diseño de redes neuronales de complejidad arbitraria. Pues simplemente con una función, gracias al algoritmo de retropropagación, se puede hacer el cálculo de todas las derivadas de forma sencilla y realizar la optimización del modelo con el algoritmo de gradiente escogido. De esta forma, con la facilidad de uso de Numpy y la función de Autograd se tiene todo lo necesario para diseñar y entrenar redes neuronales. Sin embargo, surge un nuevo problema, puesto que las redes neuronales necesitan datasets (o conjuntos de datos) de información grandes para su entrenamiento y prueba, el manejo de esta información mediante CPU es muy lento por lo que es habitual su implementación sobre un GPU.

Un hecho al momento de enfrentarse al desarrollo de una red neuronal, es que la operación que más se realiza es la multiplicación de matrices y esto representa un problema en el caso de que dichas matrices sean muy grandes. Es aquí donde surge la necesidad de hardware especializado en acelerar ese tipo de operaciones, esta tecnología no es más que las GPU. Puesto que las GPU se encuentran originalmente diseñadas para acelerar los cálculos mencionados necesarios para la renderización de escenas tridimensionales en video, entonces, son también útiles para entrenar redes neuronales, debido a que, en esencia las operaciones tanto para renderización como para entrenamiento de redes neuronales son las mismas, es por ello, que este tipo de hardware ha ganado un nuevo tipo de uso en aplicaciones de Deep Learning. Es muy importante destacar que el papel que ha tenido las GPU en el desarrollo de la eficiencia al momento de realizar estos cálculos ha sido clave para la revolución del Deep Learning y el aumento de sus aplicaciones en la última década.

Otro de los componentes clave de PyTorch mencionados en [11], es que éste posee seis herramientas clave que apoyan al desarrollo de proyectos de aprendizaje profundo. Los tensores que ya han sido señalados y sin duda componen un punto importante; Autograd que es la característica principal de este Framework, puesto que genera de manera automática un grafo computacional de gradientes y agrega algunos más. Uno de ellos es la construcción de redes neuronales, para ello existe el módulo “torch.nn”, el cual brinda capas neuronales comunes, funciones de activación y pérdida, las cuales son fundamentales para la construcción y entrenamiento de los modelos de redes neuronales. Otro de los componentes es la carga de datos, esto hace referencia a que PyTorch facilita el procesamiento y carga de información por medio de las clases “Dataset” y “DataLoader” en “torch.utils.data”, estos permiten transformar los datos personalizados en tensores y cargarlos de forma eficiente.

El entrenamiento de los modelos es otro componente clave, aquí el ciclo de entrenamiento, por lo general se implementa como un bucle tipo *for* en Python, se realiza la evaluación del modelo con los datos de entrenamiento, se calcula la pérdida y se ajustan parámetros del modelo empleando optimizadores del módulo “torch.optim”. El último componente de PyTorch es la distribución y producción, pues este framework es capaz de soportar el entrenamiento distribuido en múltiples GPU o máquinas, es-

to mediante el uso de “torch.nnparallel.DistributedDataParallel” y “torch.distributed”. Para efectuar el despliegue de modelos, PyTorch es capaz de exportar modelos empleando TorchScript o también por medio de un formato estándar como Open Neural Network Exchange (ONNX), permitiendo de esta manera su integración con varias aplicaciones y dispositivos.

Así mismo, en [11], se establece la estructura básica que posee un proyecto de PyTorch. El primer punto siempre es la “Fuente de Datos”, por lo general los datos provienen de algún tipo de almacenamiento y antes de que estos datos lleguen al modelo, necesitan un procesamiento considerable, pues estos datos deben ser convertidos en tensores que PyTorch pueda manejar empleando la clase “Dataset” en “torch.utils.data”.

El siguiente parámetro en la estructura del proyecto es la carga de datos en paralelo, para evitar la latencia de acceso, PyTorch emplea la clase “DataLoader”, la cual permite cargar datos en paralelo empleando múltiples procesos, asegurando de esta forma que los datos se encuentren listos para el bucle de entrenamiento.

El tercer punto en la estructura básica es el modelo no entrenado, aquí se lleva a cabo la construcción del modelo haciendo uso de los módulos “torch.nn”, con esto se puede construir modelos de redes neuronales, incluyendo capas que están completamente conectadas, capas convolucionales, funciones de activación y pérdida.

El bucle de entrenamiento es la siguiente sección a tratar en la estructura del proyecto, aquí en cada iteración del bucle de entrenamiento, el modelo es evaluado con los datos cargados. Seguidamente, se compara la salida del modelo con los objetivos empleando funciones de pérdida de “torch.nn”. Después se lleva a cabo el proceso de optimización del modelo, empleando los optimizadores de “torch.optim”, los parámetros de la red son ajustados para minimizar la pérdida, esto con ayuda del motor Autograd.

El quinto punto en la estructura es el entrenamiento distribuido, que sin duda se encuentra en paralelo al punto cuatro, pues en caso de modelos muy grandes, se puede hacer uso de las herramientas para un entrenamiento en paralelo, y de esta manera hay como distribuir el proceso en múltiples GPU o máquinas.

Una vez que este proceso se ha completado se tiene el modelo entrenado, de tal forma que contiene los parámetros optimizados para llevar a cabo la tarea específica que se practicó en el entrenamiento.

Como último punto en la estructura de un proyecto de PyTorch está el despliegue en producción, donde se señalan dos acciones: la primera es el despliegue del modelo, que se refiere a que el modelo se puede desplegar en un servidor, puede ser exportado a un motor en la nube o integrado en aplicaciones más grandes; la segunda es que PyTorch permite la exportación de los modelos empleando TorchScript para su ejecución independiente de Python en formato ONNX para interoperabilidad. Toda esta estructura de proyecto se puede ver en la Figura 2.20, donde de manera gráfica se muestra cómo se correlaciona cada punto de la estructura.

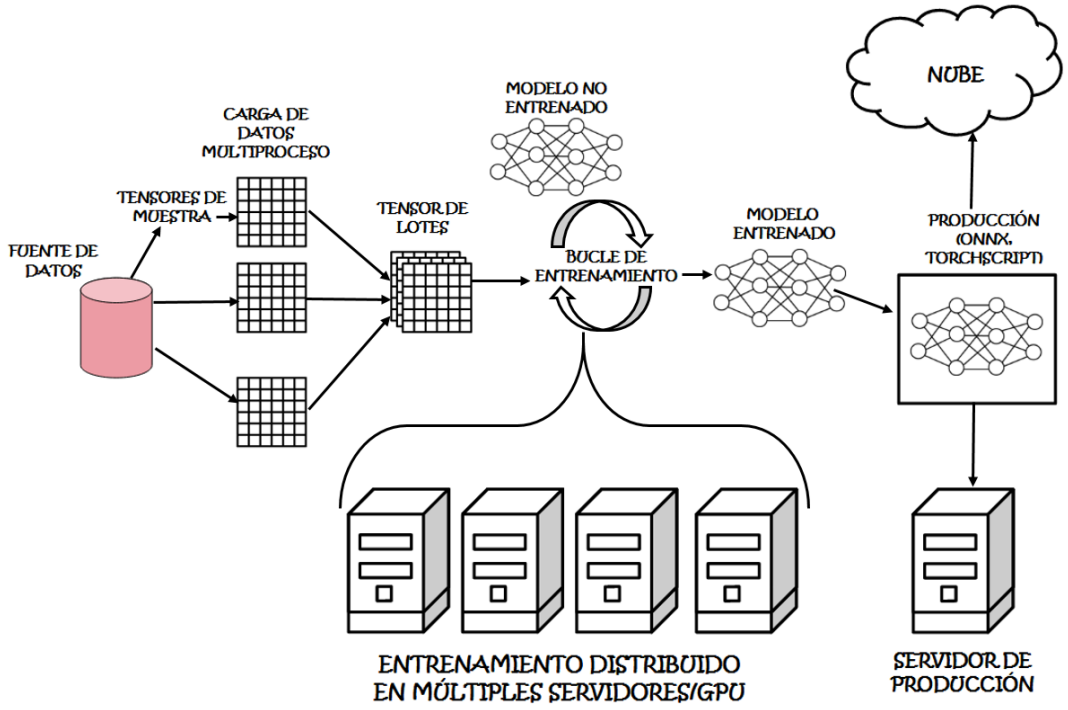

Figura 2.20: Estructura de un proyecto de PyTorch, imagen adaptada de [11].

Está claro que el Deep Learning tiene una amplia gama de aplicaciones, como la identificación de objetos, clasificación, toma de decisiones, etc. Sin embargo, para cada escenario, la sección de entrenamiento y prueba es muy diferente y es aquí donde se presenta la complejidad de lograr fusionar las comunicaciones con este tipo de técnicas.

En [58], tienen como objetivo emplear técnicas de Deep Learning para acotar un có-

digo LDPC, aprovechando que un grafo bipartido posee una estructura similar a una red neuronal artificial. La idea general de este trabajo es conseguir que las redes neuronales aprendan a decodificar un código LDPC de manera más eficiente, dado una matriz H, sin la necesidad de consumir una gran cantidad de recursos ni realizar tantas iteraciones. Específicamente en el contexto de la codificación de canal, utilizar Deep Learning presenta desafíos en consecuencia a la complejidad del entrenamiento. Por ello, la preinicialización de la estructura de la red puede ser capaz de reducir de forma significativa el tiempo de entrenamiento, tal como sugiere la técnica de Deep Unfolding. En esta propuesta se emplea redes de tipo Feed-Foward, en donde la información fluye de una capa a otra sin conexiones de retroalimentación. Para la optimización de los parámetros de la red, los autores señalan que el método más común es el descenso de gradiente, esta es una técnica iterativa para encontrar el mínimo local de una función diferenciable. El optimizador ADAM es una variación del descenso de gradiente el cuál mejora la convergencia adaptando de forma implícita el entrenamiento.

Para la definición de la red hace uso de PyTorch, como una librería adecuada gracias a sus clases y métodos específicos para la implementación. En este framework, toda red neuronal es definida mediante una clase que hereda de “nn.Module”. Esta clase contiene dos métodos principales, uno es el “Init”, aquí se definen las capas de la red, es decir, pesos y sesgos junto con sus respectivas dimensiones. También se inicializan las capas de la red considerando las dimensiones necesarias y también se puede implementar métodos de inicialización específicos.

Un segundo método, corresponde a “Forward”, también denominado propagación hacia adelante, que en general define cómo se pasan los datos a lo largo de la red desde la capa de entrada hasta la capa de salida. Así mismo, se realiza una iteración sobre las capas, aplicando funciones de actualización de nodos variables y nodos de chequeo en cada iteración para este caso particular. El punto del método Forward es caracterizar el comportamiento de entrenamiento de la red, es decir, guiar de alguna manera la interacción interna de la información para forzar el aprendizaje y reducir tiempos de entrenamiento.

En [58], hacen modificaciones respectivas para su caso de estudio, por ejemplo, en lu-

gar de emplear Neural Network (NN) convencionales, aquí se mejora un grafo existente con pesos entrenables, asignando un peso entrenable a cada uno de los mensajes empleando el grafo de Tanner suave. Así mismo, aplican la técnica de Deep Unfolding, la cual consiste en desglosar las iteraciones de un método de inferencia por en una estructura por capas análoga a una NN, consiguiendo de esta manera desatar los parámetros del modelo a través de las capas. De esta manera se expande el grafo de Tanner, interpretando las actualizaciones de nodos como capas de una NN.

Para la implementación en PyTorch, se define una matriz H de un LDPC, se genera la clase *poda* y por último la clase *Net*. En la clase *Net* es donde se definen los dos métodos principales antes mencionados con las funcionalidades respectivas. Básicamente el experimento de codificación realizado en [58], se llevó a cabo en un canal AWGN, donde empleando redes neuronales, obtuvieron una mejora en el BER a comparación de trabajos relacionados. Esto deja ver de forma directa, que el uso de PyTorch evidentemente es útil para la implementación de escenarios con redes neuronales, además, deja ver la estructura fundamental en cuanto a sintaxis para el uso efectivo de esta librería.

Así como [58], existen diversos artículos que hacen uso de PyTorch como herramienta para la implementación de redes neuronales en diferentes aplicaciones. Un ejemplo se presenta en [59], donde se investiga cómo el Deep Learning es capaz de mejorar el rendimiento en la capa física de las comunicaciones inalámbricas, en específico la estimación de canales. Dicho trabajo propone realizar un estimador de canal que emplea una red neuronal, con el fin de operar en un entorno de canal de desvanecimiento Rayleigh que varía en el tiempo. Para el desarrollo y entrenamiento del estimador hacen uso de PyTorch, y dicho estimador se desarrolla y entrena mediante datos simulados los cuales representan el comportamiento del canal de Rayleigh. De esta forma la red se ajusta para rastrear de forma dinámica el estado del canal sin la necesidad de requerir un conocimiento previo del modelo de canal o sus estadísticas.

Luego de todo el proceso de entrenamiento y prueba, los resultados indican que el estimador basado en NN posee un mejor rendimiento de Mean Squared Error (MSE) a comparación de algoritmos tradicionales y otras arquitecturas de Deep Learning. Esto implica que el estimador es capaz de prever con mayor precisión el estado del

canal.

Así mismo el estimador posee mayor robustez frente a diferentes densidades de pilotos. Lo que significa que consigue mantener su rendimiento aunque existan variaciones en la cantidad de información referencia disponible para la estimación del canal.

Finalmente, tras presentar diversos ejemplos, se observa las capacidades que posee PyTorch y justifica la elección de esta herramienta para el desarrollo de nuestro trabajo, además, al usar esta librería se logra de forma directa que el desarrollo tenga mayor visibilidad, puesto que es el Framework para implementación de redes neuronales con más uso en la actualidad, esto hará que otras personas se encuentren interesadas en comparar sus resultados con los presentados en este documento.

Capítulo 3 Metodología

3.1. Introducción y generalidades

Como se evidenció en la revisión del estado del arte presentada en el Capítulo 2, las redes neuronales han demostrado resultados prometedores en el campo de la codificación de canal frente a códigos convencionales. Sin embargo, la investigación se ha orientado mayormente a canales caracterizados por ruido aditivo Gaussiano. AWGN. A partir de este hecho, nace la motivación de llevar esta aplicación de Deep Learning a uno de los canales más conocidos y estudiados en la teoría de codificación, este es el BSC.

El proceso para llevar a cabo nuestro objetivo se basa en tomar un enfoque experimental, en el que se puede proponer diferentes modelos y arquitecturas para descubrir y obtener modelos óptimos de redes neuronales que puedan ser aplicables a un BSC en el contexto de las dificultades propias que este canal conlleva para un codificador, dada sus condiciones propias de funcionamiento y el tipo de información que puede ser introducida y obtenida del canal (bits con valores de 0 y 1).

En este sentido se han considerado dos escenarios de experimentación: El primero plantea el uso de un BSC sin retroalimentación donde el codificador y decodificador, que tienen arquitecturas de redes neuronales como eje central de funcionamiento, y se entrenarán simultáneamente; por otro lado, se contempla un segundo escenario en el que se busca evaluar las ventajas evidentes que ofrece un canal de comunicación con retroalimentación, para esto se hará uso de algunas metodologías para la generación de códigos en canales que tienen está característica. Sin embargo, llevar a cabo esto implica realizar un grupo de adaptaciones al primer escenario.

Para llevar a cabo el entrenamiento de las redes neuronales, se aplicará aprendizaje supervisado, donde a partir del dataset original se aplicará la función de pérdida de entropía binaria cruzada con la información que entregue el decodificador después de haber atravesado el canal con cierta probabilidad de error q. Luego de esto, se contrasta los resultados con respecto a códigos convencionales seleccionados para realizar la comparación de rendimiento y se busca dar una interpretación a los resultados obtenidos con el uso de los codificadores y decodificadores basados en Deep

Learning para comprender lo que la red neuronal ha descubierto.

3.2. Implementación de Códigos Convencionales

Como ya se mencionó en la Sección 3.1, la evaluación de un código descubierto mediante técnicas basadas en redes neuronales consiste en contrastar los resultados con aquellos obtenidos mediante las técnicas de codificación convencionales de mejor desempeño. Con este fin, se realizó un trabajo de investigación, con el objetivo de encontrar documentación, librerías y ejemplos que permitan implementar códigos convencionales sobre un canal binario simétrico. El propósito fundamental es el de de obtener gráficas de BER frente la probabilidad de error de un BSC, para distintas técnicas convencionales de codificación de canal. Esto posibilitará poder realizar un análisis comparativo con codificadores implementados por medio de redes neuronales convencionales o recurrentes.

En el marco de este trabajo de titulación se seleccionaron códigos clásicos y muy importantes en el área de la codificación como son los convolucionales y los códigos Hamming, así mismo, códigos didácticos y baja complejidad de implementación como los códigos de repetición y algunas técnicas de codificación de alto desempeño como son los turbo códigos y códigos polares.

Con este grupo de técnicas clásicas, se pretende evidenciar la eficiencia que pueden tener códigos generados mediante redes neuronales y analizar ventajas y desventajas de su uso. A continuación se presenta una breve revisión de cada uno de estos códigos, así como detalles de su implementación en el contexto de este proyecto.

3.2.1. Códigos convolucionales

Los códigos convolucionales, constituyen una de las aproximaciones clásicas a la protección de errores. Estos se diferencian de los códigos de bloque, debido a que el codificador es un sistema con memoria. La literatura en el contexto de la evaluación de este tipo de códigos en el contexto de BSC es escasa, por lo que se optó por la implementación de un escenario donde se haga uso de este tipo de codificadores para ver su comportamiento en el canal de interés. Con este fin, se utilizó la libre-

ría “CommPy” creada por el investigador Veeresh Taranalli, y documentada en [60]. Esta librería posee un conjunto de herramientas de código abierto, que implementan algoritmos de comunicaciones digitales en Python, empleando algunas librerías tradicionales como NumPy y SciPy. En la documentación existen múltiples ejemplos de uso sobre el proceso de codificación y decodificación con el codificador convolucional. A partir de estos ejemplos se genera el escenario de prueba descrito a continuación:

Primero se define el modelo del canal binario simétrico, mediante una función que tiene implementada la misma librería. Luego de esto, se definen los respectivos parámetros del codificador convolucional, como se detalla más adelante. Antes de eso, se debe destacar que la matriz generadora se describe matemáticamente como se indica en (3.1).

$$G(D) = \left[1 + D^2, 1 + D + D^2\right] \tag{3.1}$$

El primer parámetro que se define es el tamaño de la memoria, que para este codificador se usa dos elementos de retardo. Luego se define en el programa la matriz generadora del código convolucional. Para ello se emplea las representaciones (en base octal) de los polinomios generadores, en este caso “0o5” y “0o7”, esto se puede comprobar en [61], donde se menciona brevemente las representaciones octales de los polinomios generadores para una tasa de transmisión (rate) de 1/2. Luego de esto se genera la estructura de la trama (Trellis), aquí se usa la librería *CommPy*, ingresando la memoria del codificador y la matriz generadora en la función “cc.Trellis” de *CommPy*.

A continuación, se establece una sección con un nivel de generalidad superior: en primer lugar, se declara las probabilidades de prueba, es decir las condiciones del canal que serán evaluadas. Para este fin se define un rango de probabilidades desde 0.0001 hasta 0.15, generando 20 puntos para obtener una cantidad de datos suficiente que nos brinde una resolución de información adecuada. Acto seguido, se plantean algunas variables relacionadas a los bloques de información para el proceso de prueba.

Para la evaluación seleccionamos un número de bloque correspondiente a 100000 palabras, de la totalidad del bloque se van tomando lotes más pequeños de 100 palabras de 50 bits, es decir, el proceso de tomar un lote y pasarlo por el escenario en total se realiza 1000 veces en cada época. Para este caso, se ejecuta el experimento

por 10 épocas, y en cada una se obtiene una medición del BER promedio para cada probabilidad de error evaluada en el BSC. Finalmente, todas estas mediciones se promedian entre sí para obtener la gráfica de BER vs probabilidad con la posibilidad de adicionalmente observar sus correspondientes intervalos de confianza.

Dentro del escenario de prueba, en primer lugar, los mensajes son codificados mediante la función “conv_encode", que recibe como parámetros los 50 bits de información y la estructura Trellis, con esto se tiene como resultado 100 bits y un residuo debido al funcionamiento intrínseco de los códigos convolucionales.

Esta información codificada entra a la función de canal binario simétrico y se ve afectada en función de la probabilidad. Acto seguido, la información es recibida por el decodificador, el cual usa la función “cc.viterbi_decode” de la librería *CommPy* para la decodificación, está función, los bits que se reciben del canal, la estructura Trellis del codificador y la profundidad de traceback que recomienda el ejemplo de uso. Con esto el resultado es una palabra de 50 bits.

Para el cálculo del BER se usa la función “hamming_dist” de la librería para contar el número de errores por palabra. Esta función tiene como parámetros el mensaje original y el decodificado, de esta forma se va comparando y se van sumando los errores encontrados. Después, simplemente se genera un cociente entre el total de errores encontrados sobre el total de bits de bits de información enviados y se obtiene el BER.

En esencia, este es el funcionamiento para el escenario con uso de códigos convolucionales. Básicamente se realiza el mismo procedimiento para un codificador convolucional de 1/3. La diferencia es que la matriz generadora cambia a la que se presenta en la ecuación 3.2 y también el número de memoria a 3.

$$G\left(D\right) = \left[1 + D + D^2, 1 + D + D^2, 1 + D^2\right] \tag{3.2}$$

La representación octal de la matriz generadora para 1/3 es “0o7”, “0o7” y “0o5”, con base en esto y la memoria, se sigue el mismo proceso para conseguir la construcción del codificador y el decodificador Viterbi.

3.2.2. Turbo códigos

Otro de los códigos convencionales utilizados para comparar los resultados obtenidos con las redes neuronales corresponde a los turbo códigos. Como se menciona en la Sección 2 donde se muestra algunos resultados de su rendimiento sobre un canal AWGN, la documentación sobre su uso en un canal binario simétrico es escasa.

Para generar y comprobar el desempeño de los turbo códigos, este proyecto optó por la librería `pyturbo`, desarrollada por Paul David [62]. Esta librería incluye códigos referentes a un codificador turbo, decodificador turbo, un decodificador Trellis, un canal AWGN y también un ejemplo de su uso. Este ejemplo empieza generando un entrelazador aleatorio el cual es clave para, en base a este instanciar el codificador y decodificador, posteriormente y para cada valor de SNR se generan vectores de bits aleatorios, estos vectores pasan por el turbo codificador y después se transmiten por el canal AWGN. Finalmente, se decodifica el vector recibido, y se calcula la tasa de error de bit para cada valor de SNR.

Tomando en cuenta este ejemplo y la disponibilidad de esta librería se creó un código que se utilizará para la codificación de mensajes y su transmisión por medio de un canal binario simétrico. El proceso de este código empieza por definir funciones de cálculo de BER, la definición del modelo de un BSC y la generación de turbo codificador y decodificador. Posteriormente, se crea una función que empieza generando un bloque de 100 palabras de 50 bits cada una y un bloque de 100 palabras y 156 bits cada una (150 bits es debido a la tasa de 1/3 y 6 bits extra debido a las colas de terminación de los codificadores convolucionales).

El primer bloque genera bits aleatorios que serán transmitidos y el segundo genera bits pero con una probabilidad q de que el bit sea igual a 1 ya que esto será utilizado como ruido del canal. Posteriormente, estos datos se toman de palabra en palabra y se procede a codificar los datos de transmisión, de este modo, se tiene un vector de 156 bits, este vector, junto al vector de ruido son ingresados al canal binario simétrico.

A la salida del canal tenemos un vector de 156 bits afectado por el efecto del canal, estos se decodifican para de ese modo, tener un vector de 50 bits nuevamente. Finalmente, este se compara con el vector original y de esa forma se obtiene el valor de

BER para esa palabra con un q específico. Este proceso se hace varias veces para cada valor de probabilidad de error de bit q.

3.2.3. Código de Repetición

La lógica para codificación por repetición es sencilla, y se detalla a continuación. En primer lugar se genera la información a ser transmitida, siguiendo los mismos lineamientos mencionados para la evaluación de los códigos convolucionales. Un codificador de repetición lo que hace es leer cada bit de información y lo repite en función de un factor. Suponiendo que el factor de repetición es 3, el ejemplo que se muestra en la Figura 3.1 muestra el funcionamiento del codificador para una tasa de 1/3.

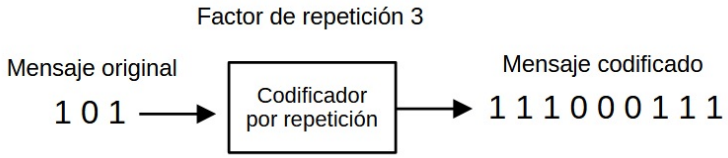

Figura 3.1: Codificador de repetición.

Como se observa en la Figura 3.1, cuando un bit ingresa al codificador, este se repite tres veces (factor de repetición 3). Por otro lado, el funcionamiento del decodificador es algo distinto, puesto que se debe señalar que este método funciona solo para factores de repetición impares, puesto que el decodificador tiene una lógica de votación. De esta manera, siempre se necesita que exista mayoría de unos o ceros, como se observa la Figura 3.2.

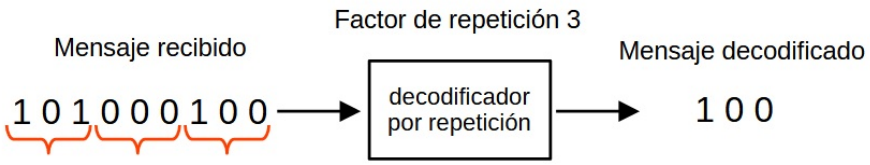

Figura 3.2: Decodificador de repetición.

Note que en la Figura 3.2, se dispone del mensaje que se envió en la Figura 3.1, pero se ha visto afectado por efectos de enviarlo mediante el canal binario simétrico. El decodificador sabe que el factor de repetición es 3, por lo que, toma tres bits de

entrada y realiza un proceso de votación. Como se observa, el decodificador logra recuperar el primer y segundo bit correctamente, sin embargo, puesto que para el tercer bit ocurre una alteración sobre los bits codificados (dos de los tres bits están complementados), entonces, no se logra recuperar correctamente.

A pesar de un funcionamiento simple que se contrasta con algunas limitaciones, este tipo de códigos son muy didácticos para realizar comparaciones con el resto de códigos.

3.2.4. Hamming

Finalmente, el último código convencional que será generado es el código de Hamming (15, 11); este código Hamming ha sido seleccionado debido a que contiene el tasa mas cercana a 3/4 y de esa forma será útil para compararlo con códigos descubiertos por una red neuronal con tasa 3/4. El proceso de codificación de este algoritmo se ha desarrollado inspirado en [34], donde se explica de manera didáctica la dinámica de los bits de paridad en un bloque de bits y cómo estos son capaces de detectar errores en un código una vez que se colocan bits de paridad. Como es ya sabido, este tipo de códigos puede corregir únicamente un error.

Para la implementación del codificador, se parte de los bits a transmitir, las posiciones en las que se colocarán estos bits y la dimensión de la matriz. Posteriormente, se colocan los bits de información en sus respectivos lugares y se realiza el cálculo de los bits de paridad de cada “porción” de la matriz. Estos bits se colocan en sus respectivas posiciones y finalmente, se devuelve un vector codificado de 15 bits.

Para la implementación del decodificador, se toman los bits que salen del canal binario simétrico y las posiciones en las que debería encontrarse la información. Mediante una implementación en bucle, se itera sobre los bits de la matriz haciendo una operación XOR entre cada una de las posiciones en las que se tiene un bit 1. Posteriormente, se verifica el resultado final de la operación XOR: en caso de ser 0, no se ha encontrado un error; por otro lado, en caso de ser distinto de 0, se ha encontrado un error. La ventaja de este código es que, en caso de existir un error, el resultado final de la operación XOR da la posición (en binario) donde se encuentra el error, de este

modo se puede corregir volteándolo. Finalmente, de las posiciones donde se debe encontrar la información, se extraen los bits y así obtenemos el vector decodificado.

3.3. Descubrimiento de códigos mediante Deep Learning

Como se mencionó en la Sección 2.5.2, el descubrimiento de códigos mediante Deep Learning se lleva a cabo en dos escenarios: el primero sobre un BSC sin retroalimentación, donde se plantean dos metodologías de construcción de mensaje codificado; y el segundo escenario donde se empleará un canal con retroalimentación, considerando que la probabilidad de error del canal hacia adelante (forward) será diferente (generalmente mucho mayor) que la del canal hacia atrás (feedback). Este enfoque permitirá explorar nuestra metodología experimental para analizar la magnitud de la influencia que puede tener el canal de retroalimentación en el desempeño final del codificador y el decodificador. Cada experimento planteado se basa en la propuesta presentada en [7] para canales AWGN, tomando en cuenta todas las adaptaciones necesarias que implica un modelo de canal binario.

3.3.1. Modelo de BSC

Uno de los primeros elementos a considerar es el modelo del canal binario simétrico. Un BSC admite únicamente valores binarios para su ingreso y salida, en particular 0 y 1. Estos valores transmitidos son afectados por el ruido del canal, lo que puede provocar que los bits se “volteen” durante la transmisión. Se le denomina canal simétrico porque la probabilidad q de que un 0 se convierta en 1 es la misma que la probabilidad de que un 1 se convierta en 0.

Para implementar el modelo de un BSC se empieza generando un vector aleatorio de bits llamado vector de ruido, el cual debe tener el mismo tamaño que el vector del mensaje. Además, el vector de ruido tiene la característica de que la probabilidad de generar un 1 en lugar de un 0 es igual a la probabilidad de que un bit se voltee. Finalmente, se realiza una operación XOR (OR “exclusivo”) entre los dos vectores, resultando en un mensaje afectado por el ruido.

Este proceso simula perfectamente los efectos de transmisión sobre un canal binario

simétrico porque al ejecutar un XOR entre un bit del vector de mensaje y un bit del vector de ruido, el bit del mensaje cambiará de estado si el bit correspondiente del vector de ruido es 1. Dado que los bits de 1 en el vector de ruido se generan según la probabilidad de volteo de bit en el canal, la salida del canal será un mensaje con una probabilidad de error en cada bit igual a la probabilidad de torsión del canal.

3.3.2. Codificación basada en Redes Neuronales

A continuación, se presenta los diferentes escenarios en lo que se evalúa el uso de redes neuronales para el descubrimiento de códigos. Inicialmente, la Sección 3.3.3 presenta el enfoque propuesto para escenarios sin retroalimentación, mientras que su incorporación al modelo de canal se presenta en la Sección 3.3.4. En total se realizarán 5 experimentos, los tres primeros experimentos serán en redes sin feedback y el resto serán en redes que usen retroalimentación. Los experimentos sin retroalimentación se diferencian en la manera que el codificador genera los datos codificados y el decodificador los toma. Por otro lado, los experimentos sin retroalimentación se diferencian en la forma en la que los datos son decodificados.

3.3.3. Codificación para un BSC sin retroalimentación

En general, puesto que en un modelo de comunicación convencional sin retroalimentación los codificadores no tienen acceso a la información disponible en el decodificador, lo conveniente es usar redes neuronales convencionales. Como se presenta más adelante, se puede utilizar modelos particulares de redes neuronales para sacar mayor provecho en caso que retroalimentación esté disponible.

En el escenario de un BSC comunicando transmisor con receptor, nuestra propuesta se basa en utilizar simplemente redes que internamente están construidas con capas lineales las cuales están interconectadas entre sí. Esto no implica que la metodología de funcionamiento de las redes sea la misma, pues se pueden realizar algunas modificaciones en relación a cómo se componen las tramas de información codificadas como se describe más adelante. En la Figura 3.3, se presenta el escenario planteado para estos experimentos sin retroalimentación.

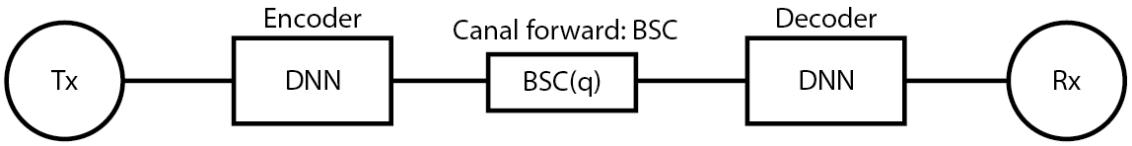

Figura 3.3: Arquitectura del escenario experimental sin retroalimentación.

Como puede observarse, la idea es que el transmisor envíe información codificada a través de un canal simétrico binario y que el decodificador realice el proceso de decodificación. A medida que progresa el proceso de entrenamiento, se espera que los pesos internos de las redes se hayan ajustado de tal manera que puedan solventar los errores producidos por el canal y se consiga un modelo descubierto por la red neuronal, y que sea capaz de obtener tasas de BER comparables con otros códigos.

A continuación describimos brevemente cada uno de los experimentos desarrollados para este escenario, fundamentalmente se distinguen en la manera en que la información de salida del codificador es generada, en el experimento 1 y 2 se emplea una tasa de codificación de 1/3, pero la forma de generar los bits para la transmisión es distinta y se explicará en las Subsecciones 3.3.3.1 y 3.3.3.2 respectivamente. Para el experimento 3.3.3.3 se emplea una tasa de codificación de 3/4, lo que induce cambios estructurales de la entrada y salida tanto para el codificador y decodificador, el detalle de estos cambios se revisarán en dicha Subsección. El tamaño de mensaje es de 50 bits codificados en binario plano para el experimento 1 y 2 y para el experimento 3 el tamaño del mensaje es de 12 bits.

3.3.3.1. Experimento 1

La estructura funcional de este experimento se inspira en la estructura propuesta en [7]. Naturalmente se realiza un conjunto de modificaciones y adaptaciones en el contexto del modelo de canal y la ausencia de retroalimentación. El proceso de obtención de un modelo entrenado con el mejor desempeño se realizaron un conjunto de pruebas, en las que se modifica una serie de parámetros que incluyen variaciones de la arquitectura de la red, y del proceso de entrenamiento sí mismo. Entre estos parámetros se puede mencionar: tamaños de lote, tasa de aprendizaje, número de épocas, tamaño de bloque, número de neuronas, probabilidades de entrenamiento y diferentes

funciones de activación. El resumen de todas estas pruebas realizadas se presenta en la Tabla 3.3, en la que también se incluye los mejores resultados obtenidos para cada una de las redes empleadas. La arquitectura del codificador utiliza la estructura que se muestra en la Figura 3.4, donde c_1 son los bits que se ingresan al codificador y $c_{1,1}$, junto a $c_{2,2}$ son los bits codificados (aquellos que salen de la red neuronal).

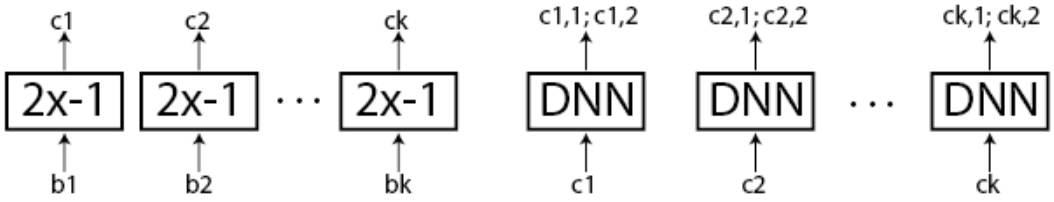

Figura 3.4: Estructura del codificador de experimento 1 (E1).

En rasgos generales, en esta estructura de codificador la trama de envío se conforma mediante un bit correspondiente al mensaje original y dos bits redundantes que se generan mediante el codificador. Este proceso se repite para cada uno de los 50 bits de cada mensaje, por lo que la palabra codificada termina con una longitud de 150 bits (tasa de 1/3). El bit original tanto para ser enviado, como para entrar en el codificador pasa por un proceso previo de señalización antipodal, donde los bits con 1 se mantienen, pero los bits con cero se convierten en -1, es decir, se realiza un proceso similar a una modulación Binary Phase Shift Keying (BPSK). Este proceso de señalización antipodal se agregó al observar que la red neuronal no reaccionaba apropiadamente cuando sus entradas tienen valores de cero y ningún modelo era capaz de recuperarse de los errores, e incluso fue peor que enviar datos sin codificar en muchos casos. La explicación a este fenómeno es que los pesos internos de las neuronas al multiplicarse por los bits de entrada marcados como cero complicaban el aprendizaje de la red.

Este proceso se realiza con cada uno de los 50 bits de mensaje como se aprecia en la Figura 3.4. Una vez se tiene la palabra codificada de 150 bits, esta se envía por el canal de comunicación el mismo que modifica la secuencia por los efectos del ruido. Un punto importante a señalar es qué para el proceso de aprendizaje automático, se debe considerar la probabilidad de error del canal BSC utilizada en entrenamiento, la cual ha sido elegida como 0.15, y que los resultados de la evaluación del modelo entrenado se obtienen probando diferentes valores de probabilidad. En particular se

realiza un barrido desde 0.0001 hasta 0.15 (el valor usado durante el entrenamiento).

La estructura del decodificador para el experimento 1 (y también para el siguiente Experimento 2) es la que se muestra en la Figura 3.5, donde $y_{1,1}$ son los bits codificados que ya han pasado por el canal.

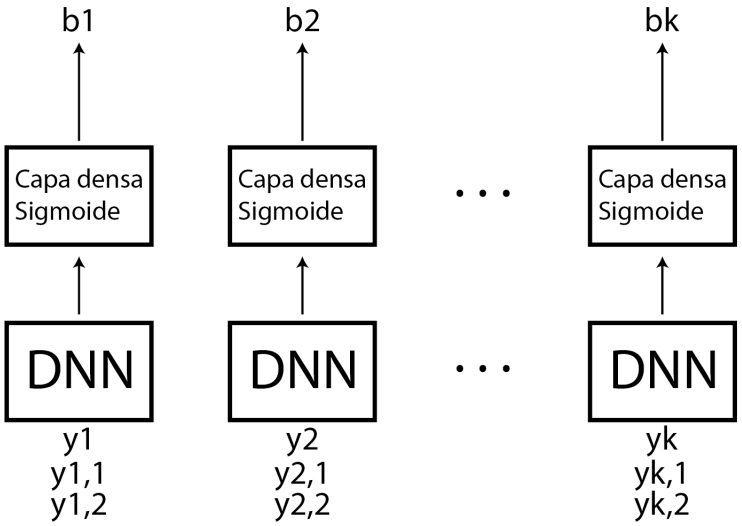

Figura 3.5: Estructura del decodificador de experimento 1 y 2.

El gráfico de arriba muestra que el decodificador toma los bits de las secuencias recibidas y los toma de tres en tres en este caso, puesto que se está trabajando con una tasa de 1/3. Estos tres bits ingresan a un par de capas lineales completamente conectadas entre sí y termina con una función de activación de tipo sigmoide. Para este punto hay que mencionar que la salida de la red tiene valores reales, esto para que la función de entropía cruzada binaria pueda calcular la pérdida y el optimizador sea capaz de ajustar los pesos correspondientes. Aun así la salida final corresponde a la salida de una función de tipo heaviside, que permite obtener valores binarios.

También hay que destacar que como parte de experimentación y por limitaciones de códigos a una sola tasa, se practicó diversas modificaciones pequeñas a la estructura para obtener una tasa de bits transmitidos (rate) diferente. Por ejemplo, para 1/2, la única diferencia a considerar con respecto a la Figura 3.4, es que el codificador ahora genera un solo bit redundante y al momento de armar la trama de envía el bit original y el de codificación obteniendo una tasa de 1/2.

Igualmente, para el decodificador, simplemente se ajusta el número de entradas, pues

ahora se reciben los bits de dos, en consecuencia, del cambio de tasa de codificación.

3.3.3.2. Experimento 2

En este experimento el objetivo es hacer que la red neuronal genere todos los bits a ser transmitidos. La estructura se muestra en la Figura 3.6.

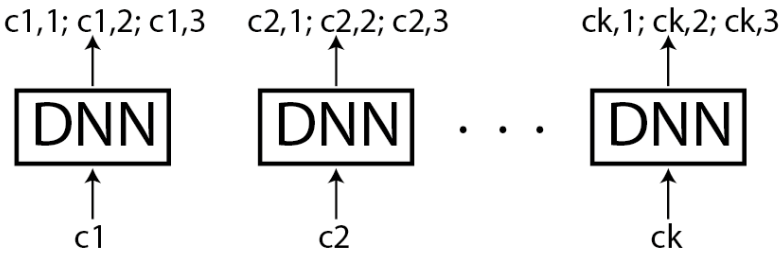

Figura 3.6: Estructura del codificador de experimento 2 (E2).

La diferencia en esta estructura es que para cada bit de información que ingresa al codificador, mismo que previamente pasa por un proceso de señalización antipodal, la red genera una salida con 3 bits codificados, es decir, el bit del mensaje original no se incluye en la trama de información codificada. Con este enfoque se busca identificar cómo construye la trama el codificador tras el proceso de aprendizaje.

Las tramas generadas por el decodificador se envían por el canal binario simétrico y llega al decodificador. La estructura que se emplea para el decodificador es la misma que se muestra en la Figura 3.5, pues ingresan los 3 bits, el cual por dentro posee 2 capas lineales conectadas entre sí y produce una salida que pasa por una función de activación sigmoide (para el cálculo de la función de pérdida) y posteriormente una función heaviside para generar la salida definitiva del decodificador.

En este experimento también se realiza una modificación para el cambio de la tasa de codificación, la consideración que se debe tener en la Figura 3.6, es que ahora la red saca 2 bits y estos son enviados por el canal. Después el decodificador también sufre un cambio en la entrada para recibir únicamente los 2 bits codificados, en general el resto de la estructura es la misma.

3.3.3.3. Experimento 3

Para el tercer y último experimento de códigos sin *feedback*, se ha cambiado ligeramente la arquitectura y la tasa de codificación usadas tanto para el codificador como para el decodificador. De la misma manera que en el primer experimento, se han realizado varias pruebas modificando parámetros como tamaños de lote, tasas de aprendizaje, número de épocas, tamaños de bloque, etc. Nuevamente, los mejores resultados se presentan en la Tabla 3.3.

La arquitectura usada para el codificador de este tercer experimento la podemos ver en la Figura 3.7.

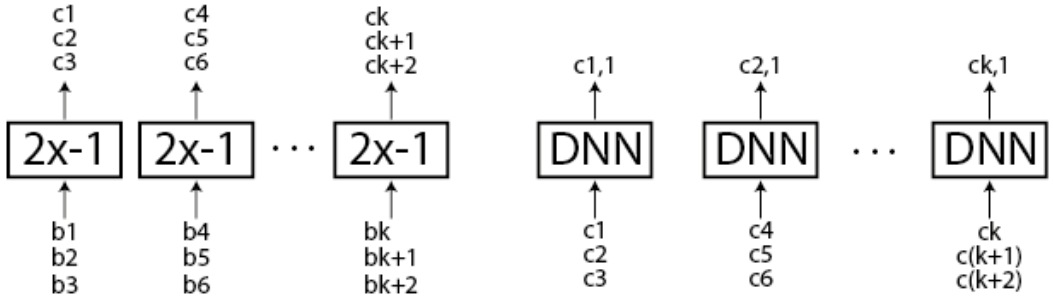

Figura 3.7: Estructura del codificador del tercer experimento (E3).

La arquitectura de este experimento se muestra de forma simplificada. Como primer paso, se transforman los bits a una representación en señalización antipodal, ya que, como se mencionó anteriormente, al existir valores de 0, estos no se ven afectados por la red neuronal y, por lo tanto, no son codificados correctamente. Una vez que se tienen los bits en forma antipodal, se arman tramas de 3 bits y se ingresan al codificador lineal, el cual, a su salida, entrega un bit de codificación. Esto se hace con todos los bits del mensaje, tomándolos en tramas de 3 bits.

Posteriormente, se concatenan los bits de codificación al final de la palabra original y se transmiten a través del canal binario simétrico, donde serán afectados por el ruido del canal. Finalmente, los bits afectados por el canal serán introducidos al decodificador, cuya estructura se presenta en la Figura 3.8.

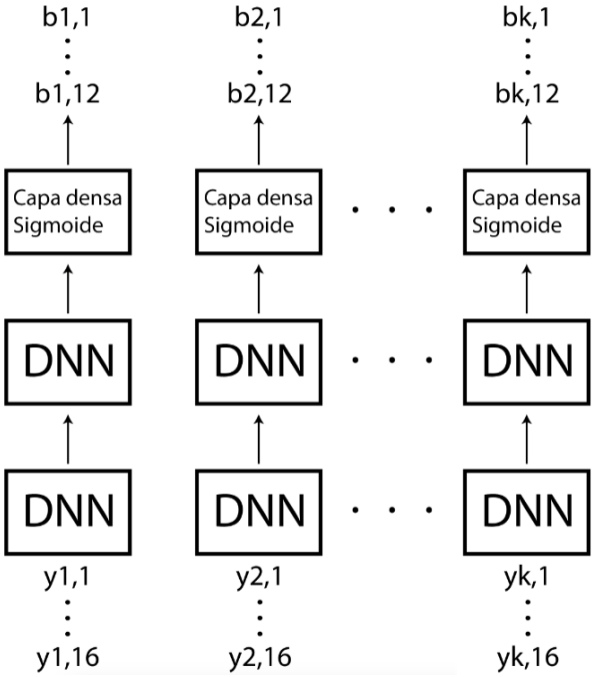

Figura 3.8: Estructura del decodificador del tercer experimento (E3).

La estructura del decodificador tiene como entrada la palabra codificada completa, es decir, los 16 bits (en caso de que se codifiquen 12 bits), y tiene como salida una palabra de 12 bits, los cuales serán tratados como el mensaje decodificado. Finalmente, este mensaje decodificado será comparado con el mensaje original para comprobar el rendimiento del codificador/decodificador. Este experimento resulta interesante debido a dos factores. El primer factor es el hecho de que forzamos al codificador a encontrar algún tipo de relación para codificar 3 bits en uno solo, y el segundo factor es que se entrega libertad al decodificador de hallar relaciones entre toda la palabra codificada y no solo en ciertas porciones para llegar a la palabra decodificada de 12 bits.

3.3.4. Incorporación de retroalimentación (feedback)

La motivación para usar canales de retroalimentación es que siempre ha sido esencial para mejorar el rendimiento de un canal de comunicación. La disponibilidad de un canal para retroalimentación brinda la posibilidad de contar con información sobre el estado de la transmisión de datos, esto ayuda a ajustar y optimizar algunos parámetros los cuales ayudan a garantizar una comunicación eficiente y confiable. Existen

algunas maneras para hacer uso del canal de retroalimentación, pero la idea general siempre es que este aporte información acerca del canal de comunicación y realizar mejoras en codificación que permitan aumentar mejorar el BER del canal.

Al realizar codificación con retroalimentación, es decir, codificación que cuenta con acceso a la información que llegó al decodificador, es necesario aprovechar esta información de la menor manera. Para ello se hace uso de redes neuronales recurrentes tanto en codificador como en decodificador. Note que es importante recalcar que el canal de retroalimentación en la práctica es ruidoso (aunque un feedback perfecto puede considerarse también para la determinación de los límites teóricos de la comunicación). El nivel de ruido en el canal de retro alimentación también es relevante, puesto que de éste depende si el canal puede en realidad contribuir a disminuir los valores de BER o en el peor de los casos exacerbarlos, lo cual es motivo de interés académico y continua investigación.

Usando la retroalimentación de la transmisión para alimentar a las redes neuronales se han generado más experimentos los cuales se asemejan a los realizados en la Sección 3.3.3. De esta manera se puede llevar a cabo una comparación entre los códigos obtenidos con y sin retroalimentación y así evaluar la influencia de la retroalimentación.

La Figura 3.9 muestra un esquema simplificado de la estructura utilizada, en ella se puede observar tanto el canal usado para la transmisión de datos como el canal usado para la retroalimentación.

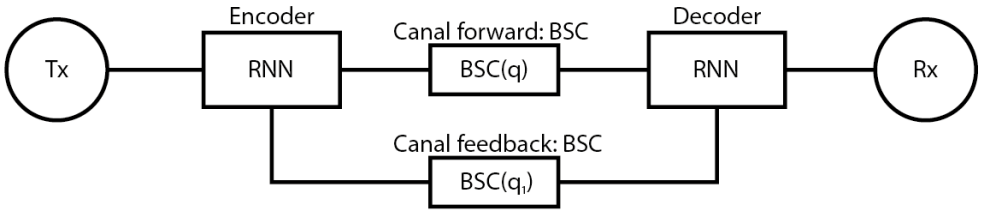

Figura 3.9: Arquitectura del escenario experimental con retroalimentación.

3.3.4.1. Experimento 4

En la Figura 3.10, se presenta la estructura del codificador para este escenario con retroalimentación. La idea tal como se observa es que tenga dos fases, en la primera

se envía el bit de información del mensaje original, teniendo a consideración que tanto el canal hacia adelante como el de retroalimentación no experimentan ningún retardo, el codificador tiene acceso a feedback instantáneo. En otras palabras, en la siguiente ronda (uso del canal $i+1$) es posible que el codificador disponga del valor que entrega el canal de retroalimentación para cada bit transmitido en el uso de canal i.

Existe un problema evidente para el caso del primer bit a ser codificado, y del cual no se tiene feedback, por lo que las entradas de la red recurrente para este se completan con ceros. A partir del segundo bit, ya se dispone de la retroalimentación de los bits de codificación empleados en la ronda anterior. De esta manera, al codificador ingresan tanto el bit de información, como la retroalimentación de ese bit y también la retroalimentación de los bits de codificación anteriores, donde el bit de información se denota como c_1, la retroalimentación de ese bit como $\tilde{y}_1$ y la retroalimentación de los bits de codificación anteriores como $\tilde{y}_{1,1}$ y $\tilde{y}_{1,2}$. Esto se refleja en la Figura 3.10:

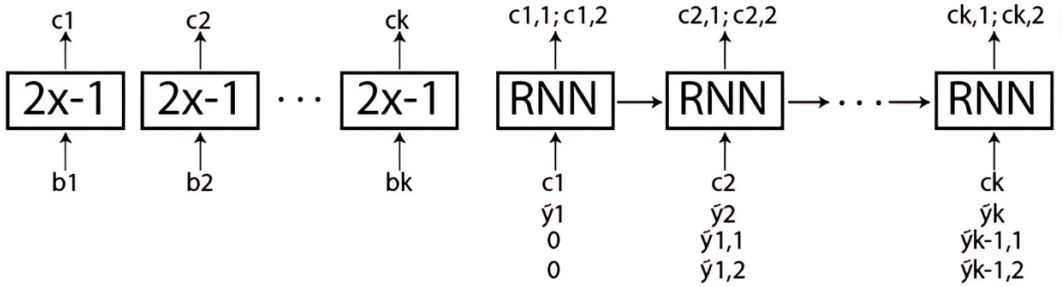

Figura 3.10: Estructura del codificador de experimento 4 (E4).

Al igual que en los otros experimentos se aplica el proceso de cambio a señalización antipodal para las señales que ingresan al codificador se ejecuta de manera similar por la misma razón que se explicó con anterioridad. El decodificador tiene una estructura que se presenta en la Figura 3.11.

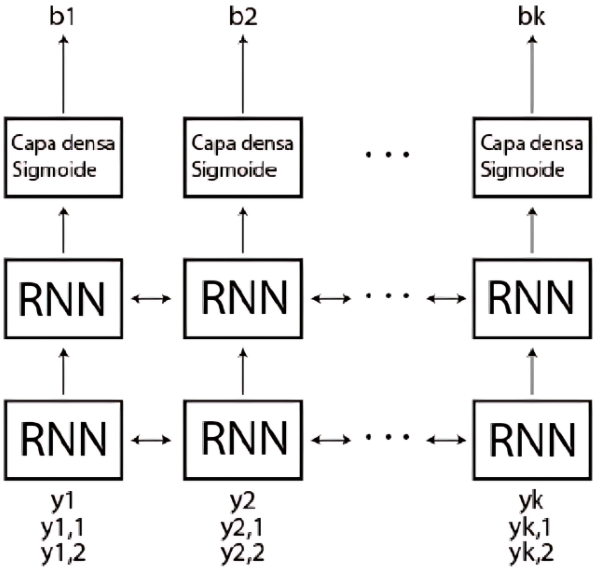

Figura 3.11: Estructura del decodificador de experimento 4 (E4).

Note en la Figura 3.11 que al igual que en caso sin retroalimentación, el decodificador toma los bits de entrada en grupos de tres, puesto que se emplea una tasa de 1/3. Estos bits ingresan en múltiples capas construidas con neuronas recurrentes de tipo GRU. El objetivo de este enfoque es que estas capas vayan guardando la información de procesos anteriores (aprovechando la retroalimentación) y empleen esa información para llevar a cabo un mejor proceso de decodificación.

Después que la información pasa por la capa de neuronas recurrentes entra a una capa lineal densa donde la salida que produce resulta del uso de una función de activación sigmoide son valores cercanos a cero o uno. Es muy importante entender que el hecho de que la salida no sea binaria no afecta en realidad al escenario de canal binario simétrico que planteamos, puesto que esa información de salida jamás vuelve a entrar al canal de vuelta, por lo que se mantiene concordancia con el objetivo principal del experimento, que es evaluar el comportamiento de las redes en este tipo de canales. Es así que las señales atraviesan una función de salida dura (heaviside) antes de ingresar a los canales hacia adelante (forward) y de retroalimentación (feedback).

También debe considerarse que en el decodificador la capa de red recurrente es bidireccional, esto implica que se usan dos capas GRU en cada paso de tiempo, una de ellas procesa la secuencia de entrada en la dirección temporal normal y la otra procesa en la dirección opuesta. También se debe tener en cuenta que el bias se encuentra

activado, esto permite que la función de activación no se encuentre anclada al origen, por lo que es capaz de ajustarse mejor a los datos.

En este experimento, también se lleva a cabo modificaciones para obtener una tasa de 1/2 con el objetivo de contrastar los resultados con códigos convencionales. Con base en la Figura 3.10, se debe considerar que solo se tienen 3 entradas, una es el bit del mensaje original, otra corresponde a su retroalimentación y finalmente, la retroalimentación de un bit redundante codificado. La salida del codificador entrega un solo bit de codificación. La palabra creada por el codificador consiste entonces del bit original y el codificado, de esta manera, el decodificador recibe 2 bits y entrega como resultado un solo bit decodificado.

Así mismo se realizan cambios en cuanto a la estructura de las redes, pues revisando el estado del arte, se pudo apreciar que las redes LSTM tienen una mayor cantidad de almacenamiento interno en sus capas, por lo que se nos hizo interesante realizar las mismas pruebas pero con redes que tengan este tipo de capas. Para realizar esto, simplemente se llevó a cabo el cambio respectivo de las redes con uso de la librería.

De igual forma, se realiza el cambio para probar a tasas de codificación 1/2 y 1/3, la forma de realizarlo es exactamente la misma que ya se describió para RNN.

3.3.4.2. Experimento 5

Para el quinto experimento se ha utilizado una arquitectura similar a la vista en el experimento 3. Esta también tiene una tasa de codificación 3/4, es decir que por cada 3 bits de información, se tiene uno de codificación. El esquema de la arquitectura usada para el codificador de este experimento la podemos ver en la Figura 3.12.

Se puede divisar de forma sencilla cómo se conforma la red que sirve como codificador. En primer lugar y de forma análoga al experimento 3, se transforman los bits a una presentación antipodal para un mejor funcionamiento de la red. Posteriormente se arman tramas de 7 bits los cuales incluyen 3 bits de información, la retroalimentación recibida al enviar esos 3 bits y por último la retroalimentación correspondiente a la transmisión del bit de codificación. La salida de este codificador es 1 bit de codificación para así mantener la tasa (rate) de 3/4.

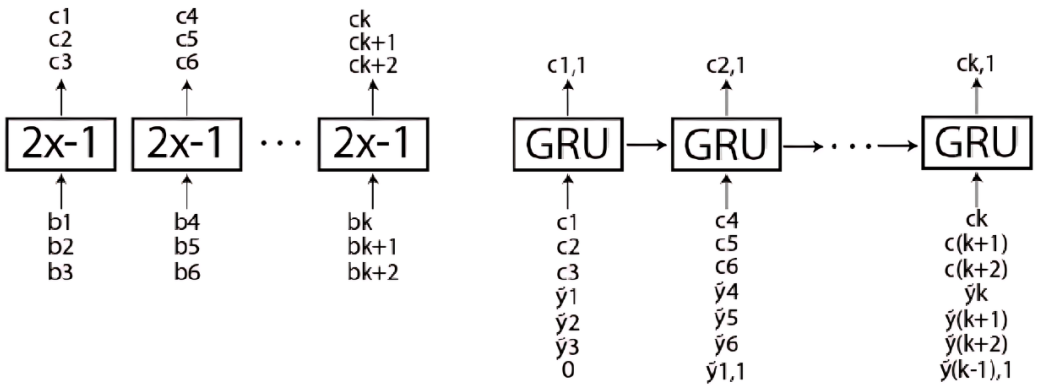

Figura 3.12: Estructura del codificador del quinto experimento (E5).

Después de codificar la información se concatenan los bits de codificación al final de la palabra original y así se transmiten a por medio del canal de comunicación. Finalmente, los bits afectados por el ruido del canal serán introducidos al decodificador, cuya estructura se presenta en la Figura 3.13.

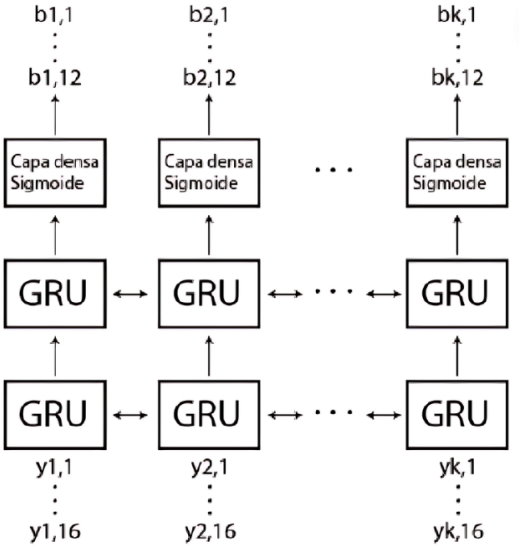

Figura 3.13: Estructura del decodificador del quinto experimento.

Este decodificador tiene como entrada todos los bits que salen del canal, es decir, la palabra codificada completa y como salida entrega una palabra de 12 bits. Estos 12 bits serán tratados como el mensaje decodificado. Finalmente y de forma similar al resto de experimentos, este mensaje decodificado será comparado con el mensaje original para comprobar el rendimiento del codificador/decodificador. Este experimento resulta interesante debido a que además de los dos factores descritos en el experimento 3, se puede observar si el canal de retroalimentación tiene influencia en la

codificación.

3.3.5. Entrenamiento de los modelos

El proceso de entrenamiento en general de las redes planteadas en cada uno de los cinco experimentos explicados anteriormente se ejecuta de manera similar y siguiendo ciertos lineamientos. Como se menciona en el Capítulo 2, especialmente en la Sección 2.5.5, existe un proceso estándar que se debe considerar para entrenar redes neuronales de manera exitosa.

En primer lugar, se debe considerar la fuente de los datos, cuyo proceso de construcción de la información se muestra en la Figura 3.14.

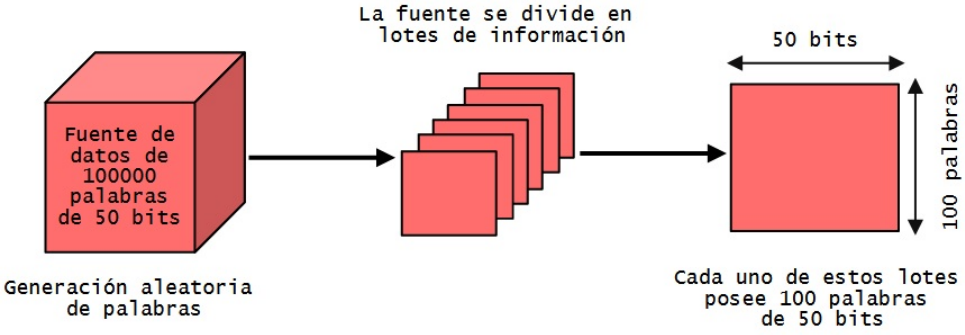

Figura 3.14: Construcción de la información.

La fuente de información se obtiene generando aleatoriamente 100000 mensajes codificados en binario plano correspondiente a palabras de 50 bits (sin embargo, se debe considerar que la única diferencia con tasa de codificación 3/4, es que el tamaño de las palabras es de 12 bits). La idea general es dividir ese bloque de información en lotes de 100 palabras de 50 bits, lo que da un total de 1000 lotes que poseen 100 palabras de 50 bits.

Se tiene que considerar que, de igual manera, que antes de ingresar al entrenamiento, debe crearse una fuente de información de ruido en concordancia con la tasa de codificación que se esté trabajando. Los lotes de ruido se generan de manera aleatoria usando una distribución de probabilidad Bernoulli, con una probabilidad de error q, correspondiente a la probabilidad de entrenamiento. Básicamente, se sigue la misma idea ilustrada en la Figura 3.14, con la diferencia que se usa una distribución Bernoulli

para la generación de ruido en cada bit de cada una de las palabras. Este lote de información de ruido se usa para modelar el canal como un simple modelo aditivo en módulo-2.

Cada ronda de entrenamiento utiliza uno de estos lotes, cuando termina el lote se realiza el cálculo de la función de pérdida, específicamente, la entropía binaria cruzada, la cual es ampliamente empleada para redes neuronales que poseen salidas sigmoidales como este caso. El uso de esta función da como resultado un valor, el cuál ingresa al optimizador *Adam*, correspondiente a un algoritmo de optimización de gradiente estocástico usado para el entrenamiento de redes neuronales profundas. Este optimizador mezcla dos métodos de optimización, el AdaGrad y RMSProp [63], brindando de esta forma un método eficiente para ajustar parámetros del modelo. El último paso del entrenamiento, consiste en reajustar los pesos. Este proceso se repite hasta terminar con todos los lotes y posteriormente con cada una de las épocas de entrenamiento en la que se ingresa un nuevo bloque de datos. En la Figura 3.15, se puede apreciar de manera gráfica este proceso.

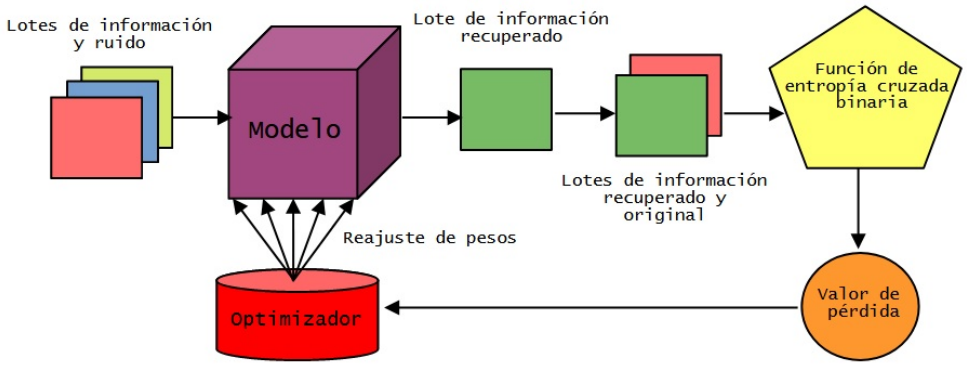

Figura 3.15: Entrenamiento y ajuste de pesos.

Se debe enfatizar que se realiza un entrenamiento simultáneo del codificador y decodificador. El modelo se planteó de esta manera ya que tomamos como inspiración la metodología de entrenamiento empleada en [7] en el que fundamentalmente se sigue la metodología de los auto-codificadores. Además que el codificador es dependiente de la información que obtiene el decodificador, existe una correlación implícita entre las dos estructuras de redes neuronales lo que obliga a una optimización simultánea.

3.3.6. Evaluación de los modelos.

Para poner a prueba los modelos obtenidos tras el entrenamiento de las redes neuronales, se realiza un proceso similar a aquel usado para la optimización en cuanto al tratamiento de la información. La principal diferencia con el periodo de entrenamiento, es que ya no se realiza ninguna clase de reajuste de pesos a las estructuras internas, sino simplemente evaluar la salida para un determinado valor de entrada y estimar cuántos bits experimentaron errores de decodificación. En la Figura 3.16, se presenta la arquitectura de prueba para el modelo.

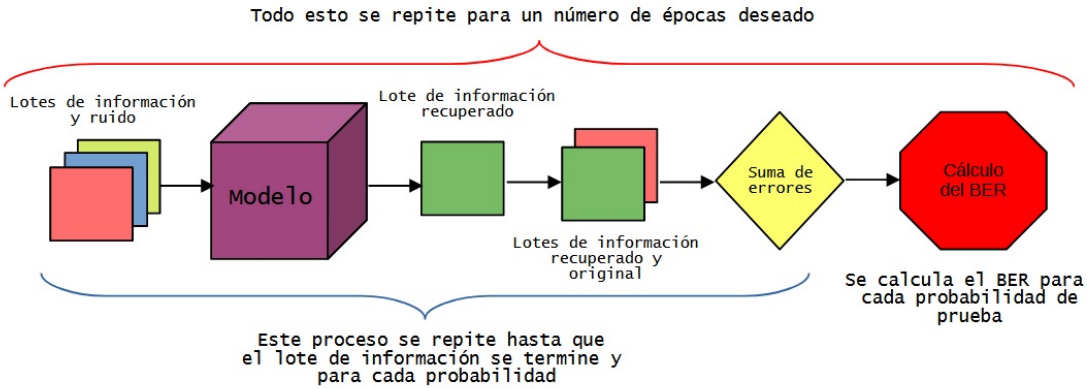

Figura 3.16: Proceso de prueba del modelo entrenado.

El cálculo del BER es uno de los puntos más importantes, puesto que se constituye como una de las mejores métricas para comparar el desempeño que tiene el modelo para corregir errores. Para ello, se debe entender la cantidad de datos que se comparan y cómo se calcula.

En primer lugar, sabemos tenemos definida una variable “num block”, que indica el número de palabras en la fuente de información, en este caso 100000. El “num block” se divide en lotes con un número de palabras denominado “batch size”, que en los escenarios planteados es de 100. Es decir, la cantidad de lotes existentes es de “num block/batch size”.

Cada lote posee un número de palabras dado por “batch size”, como ya se mencionó y cada mensaje posee 50 bits. Por ende, cuando se suman los errores se sigue este proceso. Se compara con el lote original y se marcan los bits con error como 1. Luego

se suman los 50 bits de cada palabra, esto da como resultado un arreglo con 100 filas y una columna con la suma de errores de cada palabra.

Luego se suman los errores de todas las palabras, obteniendo de esta forma un valor de errores por lote. Este valor se va sumando a una variable que almacena el valor de error de cada uno de los lotes de prueba. De esta manera se obtienen los errores totales para una probabilidad. Por lo que el BER para esa probabilidad esta dado como muestra en (3.3).

$$\text{BER} = \frac{\text{Total de errores}}{(\text{block len}) \cdot (\text{batch size}) \cdot \left(\frac{\text{num block}}{\text{batch size}}\right)} \tag{3.3}$$

Lo que se simplifica a (3.4).

$$\text{BER} = \frac{\text{Total de errores}}{(\text{block len}) \cdot (\text{num block})} \tag{3.4}$$

La ecuación (3.4), indica que la multiplicación de la longitud del bloque, por el número de palabras totales en la fuente de información no es más que la totalidad de los bits de información enviados, por ello, la función de BER empleada puede ser reescrita como se presenta en (3.5).

$$\text{BER} = \frac{\text{Total de errores}}{\text{Total de bits enviados}} \tag{3.5}$$

Claramente, la ecuación (3.5) corresponde a la definición del BER, que es el número de bits erróneos, sobre el número total de bits que se transmitieron. De esta manera, aseguramos que las medidas de BER que se obtengan en los resultados tienen coherencia respecto al proceso que se ha planteado.

3.3.7. Ajuste paramétrico de los modelos.

Este proceso, que se lleva a cabo de manera experimental está orientado a afinar el modelo para obtener los mejores resultados. A partir de los modelos de red propuestos para cada experimento se procede a la modificación de parámetros hasta obtener la combinación con mejores resultados.

Previo a la modificación de parámetros, se tuvo que tomar algunas decisiones. La primera de ellas fue la selección de la probabilidad de error en el BSC que se utiliza durante el entrenamiento, así como también aquellas probabilidades del BSC utilizado para las pruebas. En este contexto, nos basamos en el artículo [5], Figura 2.11, en la que el intervalo de prueba en canales binarios simétricos, tiene como máximo 0.15 de probabilidad de error q en el BSC. De esta manera creamos un marco de referencia para poder ejecutar el contraste de resultados. Con base en este antecedente se realizan múltiples pruebas con diferentes probabilidades de error en el BSC durante el entrenamiento, registrando los resultados con el fin de determinar la más adecuada. En este procedimiento se asume que la probabilidad de error del canal no debería ser muy alta como 0.15 porque existirían demasiado errores o muy baja como 0.0001 porque no habría una buena cantidad de errores a corregir. En la Tabla 3.1, se presenta el resultado obtenido de probar diferentes probabilidades de entrenamiento.

Tabla 3.1: Análisis de mejor probabilidad de entrenamiento

Probabilidades de entrenamiento	**Tasa de éxito de entrenamiento**
0.0008	20 %
0.001	25 %
0.0015	35 %
0.005	10 %
0.01	10 %

En la Tabla 3.1, se menciona el término tasa de éxito de entrenamiento, en este contexto hace referencia al porcentaje de las 20 veces que se intentó entrenar la red con cada probabilidad y se logró entrenar dicho modelo, por ejemplo, tomando el caso de probabilidad de error q 0.0015, este posee 35 % como tasa de éxito de entrenamiento, lo que quiere decir, que de 20 veces que se intentó entrenar la red, en 7 ocasiones la red presentó signos positivos sobre su entrenamiento y no un estancamiento en un determinado valor de pérdida.

Para obtener estas tasas de éxito de entrenamiento en función de la probabilidad, se realizó un entrenamiento repetido 20 veces en la red neuronal propuesta en el experimento 1, luego se cuenta las veces que el modelo muestra signos donde el entrenamiento está convergiendo a un resultado exitoso, siendo uno de estos signos

que el BER disminuya y también analizando que la pérdida no se mantenga oscilando en algún punto del experimento. Cada vez que se notaba una mejora en la función de pérdida se contaba como un entrenamiento exitoso. Con base en esto, la probabilidad de entrenamiento con mejor resultado fue 0.0015, ya que presenta la mayor tasa de éxito de entrenamiento.

Puesto que se observó una tasa de éxito de entrenamiento no tan positiva, entonces, no puede tomarse como único parámetro de ajuste, también se consideró los valores iniciales de los pesos con los que comenzaba el modelo. Por ello, el siguiente paso a tomar fue guardar los valores de los pesos iniciales cuando existía un buen entrenamiento, lo que nos llevó a identificar experimentalmente, valores de pesos iniciales para los experimentos 1 y 2 a tasa 1/3. Al guardar esos pesos, la tasa de éxito de entrenamiento subió al 100 % para esos casos. Es decir, al cargar esos pesos como iniciales a esos modelos, siempre lograba entrenarse de forma correcta.

Para el resto de modelos, los valores iniciales de los pesos resultan irrelevantes, por lo que simplemente se adopta como el mejor valor de probabilidad para entrenar la red que es 0.0015.

Por otra parte, el intervalo de probabilidades de error para el canal en el proceso de evaluación los modelos, se eligió considerando como mínima probabilidad 0.0001 y como máxima 0.15, esto en base a los artículos revisados como [5] en los que se presenta resultados comparables para BSC. Además, el proceso experimental demostró que al utilizar probabilidades de error en el canal más altas ocasiona que el canal sea intratable y por ende la información que se capturaba ya no era de interés. De esta manera, se procedió a las pruebas de los diferentes parámetros, los valores usados para cada parámetro se resumen en la Tabla 3.2.

Note que los valores de prueba para cada parámetro mostrados en la Tabla 3.2 se realizaron en cada una de las redes planteadas en los experimentos. Se realizaron 10 repeticiones con cada valor de cada uno de los parámetros. Luego en base al BER resultante para el intervalo de prueba se promedia las 10 salidas que se obtuvieron con cada uno de los valores de los parámetros y se fue clasificando por categoría los resultados. Por ejemplo, en función del promedio del BER que se obtuvo con un “batch size"de 25, 50, 100 y 200, se analizó en cuál se tuvo mejores resultados y

Tabla 3.2: Parámetros y valores de prueba.

Parámetros	Valores de prueba					
Batch size	25	50	100	200	-	-
Num Epoch	10	50	100	500	1000	2000
Num Block	500	5000	10000	25000	50000	100000
Learning rate	0.01	0.001	0.00085	0.0001	-	-
Num neuronas	100	300	500	-	-	-
Tasa de codificación	1/2	1/3	1/5	1/7	1/9	3/4
Num Capas	2	3	10	25	50	-

el valor con mejor rendimiento fue el que se seleccionó de manera definitiva para entrenar al modelo final. En base a esta explicación, en la Tabla 3.3, se presenta los mejores parámetros obtenidos para el entrenamiento de los modelos.

Tabla 3.3: Mejores valores de los parámetros para entrenamiento de los modelos.

Parámetro	Valores óptimos
Batch size	100
Num Epoch	Para todos los experimentos: 10 A excepción de experimento 4 con : 100
Num Block	100000
Learning Rate	0.00085
Num Neuronas	100
Tasa de codificación	1/2, 1/3 ó 3/4
Num Capas RNN	Para experimentos 4 y 5 : 2

En función de los resultados presentados en la Tabla 3.3 se deduce lo siguiente: el valor de *batch size* o tamaño de lote es el mismo para todos los modelos debido a que, durante el proceso de experimentación, al evaluar distintos valores de prueba no existió una diferencia sustancial en los valores de BER que pueda sugerir una inclinación hacia un valor determinado para este parámetro, por lo que se seleccionó el valor de 100. Es importante mencionar que en los experimentos presentados en [7], el *batch size* es de 200, mientras que en [8] es de 2000, sin embargo, sus resultados demuestran que el *batch size* no es un parámetro que influya significativamente el desempeño de los modelos.

Por otro lado, el parámetro *Num Epoch*, se seleccionó de manera similar para todos

los experimentos a excepción del experimento 4. Esta diferencia surge como consecuencia del tipo de capas que posee dicho experimento, donde se hace uso de las RNN que pueden ser de tipo GRU o LSTM. En estas redes más complejas, al poseer capas ocultas y funciones de memoria a corto y largo plazo, se asume que se necesitan más muestras para alcanzar una mejora en el rendimiento general del modelo. Sin embargo, incrementar el número de épocas no implica que en todos los casos que se obtendrá un mejor resultado y finalmente la elección de este parámetro nace netamente de un trabajo de prueba y error hasta alcanzar un valor adecuado experimentalmente. Para el resto de casos, este parámetro se mantuvo dado que se observó experimentalmente que el desempeño del modelo no mejoró sustancialmente para un número mayor a 10 épocas de entrenamiento.

Para el parámetro de *Num Block*, se proyectaba de que el mejor valor para este parámetro debía ser alto, esto debido a lo revisado en [8], donde indican que el tamaño del mensaje deja de ser relevante para lotes de entrenamiento más grande, además bajo la asunción de que mensajes cortos no producen buenos códigos como se afirma en [8]. Este hecho pudo verificarse en el proceso experimental. Cabe destacar que entre valores seleccionados de 50000 y 100000, no existió mayor diferencia en el rendimiento del modelo, sin embargo, se optó por utilizar el valor de 100000, esperando que posteriormente en la etapa de entrenamiento, y en combinación con otros parámetros, el resultado que se obtenga mejore a consecuencia de tener más muestras de entrenamiento.

El parámetro de *Learning Rate* es un parámetro crítico para el entrenamiento de un modelo, pues si se elige uno muy alto, la fluctuación entre los pesos de la red cuando se ajusten será muy grande y esto podría resultar en que nunca se alcance el valor óptimo de los pesos. Si el valor de este parámetro es muy pequeño, la fluctuación ya no será tan grande, sin embargo, el tiempo para alcanzar el valor óptimo será mayor. Por ello, la elección de este parámetro también nace netamente de una acción experimental. Para cada experimento el valor de este parámetro que resultó en un mejor desempeño de los modelos fue 0.00085, esto debido a que, al probar con los valores más grandes mostrados en la Tabla 3.2, la redes no presentaron mejoría, a excepción del valor 0.001, donde ocasionalmente sí se obtuvo una mejora. Por otro lado, al pro-

bar con el valor más pequeño (0.0001), las redes se entrenaron exitosamente, pero el tiempo y el número de muestras necesarias para alcanzar una mejoría incrementó notablemente. A partir de ello, una evaluación experimental entre valores de 0.001 a 0.0001, se alcanzó el valor de 0.00085, para el cuál se hicieron las respectivas 10 repeticiones para descartar cualquier tipo casualidad y se eligió como el valor óptimo para el entrenamiento de las redes.

En relación al número de neuronas, se debe dos puntos relevantes: en primer lugar, este parámetro se refiere al número de neuronas en la capa intermedia en los dos modelos, tanto para el codificador y como para el decodificador; en segundo lugar, se tiene la selección de 100 neuronas para todos los experimentos como un resultado experimental, tras evaluar los modelos con 300 y 500 neuronas. Estos valores no produjeron alguna mejoría significativa, pero sí un aumento de la carga computacional para el entrenamiento de los modelos, incrementando considerablemente los tiempos de ejecución de los experimentos. Lo propio ocurre en relación al número de capas ocultas, en particular para el caso de los experimentos 4 y 5 que poseen retroalimentación, pues al probar con un mayor número de capas no se apreciaron mejorías y el uso de más recursos produjo un mayor tiempo de entrenamiento que no se tradujo en mejores resultados de los modelos.

Finalmente, se debe mencionar que la tasa de codificación (rate) no constituye en un parámetro de afinación de los modelos sino más bien, que se escoge en función del interés práctico y de resultados previos disponibles para códigos convencionales. A consecuencia de que la implementación de códigos convencionales se ejecutó exitosamente para tasas de 1/2, 1/3 y 3/4, se optó por evaluar estas mismas tasas para los modelos neuronales. Note que estas tasas se emplean habitualmente en diferentes trabajos para evaluar diferentes códigos.

3.4. Procesamiento y generación de resultados

Para poder comprobar y analizar el desempeño de las redes neuronales propuestas en los experimentos mencionados, se crean gráficas que muestran el BER obtenido para diferentes valores de la probabilidad de error del canal q. Dado que se realizan

numerosas simulaciones es necesario calcular la media aritmética de todos los valores de BER obtenidos para cada una de los valores de probabilidad de error del canal evaluada. Adicionalmente, se calcula su intervalo de confianza que se basa en la distribución t de Student, y se obtiene inicialmente el Standard Error of the Mean (SEM), definido como:

$$\text{SEM} = \frac{s}{\sqrt{n}} \tag{3.6}$$

donde s es la desviación estándar muestral y n es el tamaño de la muestra. Posteriormente, con el valor de SEM, se calcula el intervalo de confianza utilizando la expresión:

$$\bar{x} \pm t_{\alpha/2,\,df}\left(\frac{s}{\sqrt{n}}\right) \tag{3.7}$$

Aquí, $\bar{x}$ representa la media muestral, $t_{\alpha/2,\,df}$ es el valor crítico de la distribución t de Student para un nivel de confianza del 95 % y df son los grados de libertad. s es el error estándar de la media y n es el tamaño de la muestra.

Una vez obtenidos estos datos, se genera un archivo de tipo ".xlsx" que contiene tres columnas: la primera con los valores de probabilidad de error, la segunda con los valores medios de BER y la tercera con los valores del intervalo de confianza. Este archivo se utiliza finalmente, para generar las figuras. Estos archivos se procesan en MATLAB, donde se gráfica BER vs q configurando el eje donde se encuentran los valores de BER como logarítmico.

3.5. Herramientas y Entorno de Desarrollo

Para desarrollar los experimentos aquí descritos, ha sido necesario utilizar herramientas que faciliten y agilicen los tiempos de ejecución de los programas. Una de las dificultades comunes al entrenar redes neuronales es enfrentan tiempos de entrenamiento excesivamente grandes. Este fenómeno se debe principalmente a que estas porciones de código realizan de manera iterativa miles de operaciones matriciales, lo que genera una carga computacional bastante alta para un computador común. Como consecuencia de esta alta carga computacional, los tiempos de ejecución son extremadamente elevados, llegando algunos a ser de varias horas e incluso días, dependiendo de los parámetros de simulación. En este punto, hemos contado con dos

aliados:

El primero de ellos es la librería utilizada para la generación de las redes neuronales, *PyTorch*, que incluye compatibilidad con la herramienta Compute Unified Device Architecture (CUDA), la cual permite ejecutar código en la GPU. CUDA es sumamente útil debido a que las unidades de procesamiento gráfico están diseñadas para realizar operaciones matriciales de gran magnitud en cortos periodos de tiempo, ya que están pensadas para procesamiento de video principalmente, como por ejemplo videojuegos. Obviamente, esto hace que los tiempos de entrenamiento y prueba de la red neuronal sean mucho más rápidos.

El segundo aliado corresponde a la plataforma Google Colab. Esta plataforma ha permitido disponer de un servidor para ejecutar el código de las redes neuronales por un módico pago. Este servidor cuenta con una CPU potente; sin embargo, lo más importante ha sido las características de la GPU con la que cuenta. El entorno de ejecución de GPU tiene gran potencia y, al pagar la suscripción, se dispone de un modo de memoria Random Access Memory (RAM) ampliada, lo que acelera aún más la ejecución de los códigos. Esto ha hecho posible que los tiempos de entrenamiento bajen al punto de permitir experimentar con varios parámetros en las redes, obteniendo las respuestas en tiempos de entrenamiento relativamente cortos, como se presenta en la siguiente Subsección que detalla los tiempos de entrenamiento medidos utilizando la herramienta Colab para diferentes experimentos.

3.5.1. Tiempos de ejecución en la plataforma Google Colab

Esta sección se enfoca en la medición de los tiempos de ejecución de los distintos experimentos presentados en este trabajo. En particular, se utiliza la plataforma Google Colab [1] para todos las ejecuciones, asegurándose que todas cuenten con las mismas condiciones de hardware. Los tiempos de entrenamiento de los experimentos sin feedback se presentan en la Figura 3.17, que muestra cómo los experimentos con una tasa de codificación menor toman una mayor cantidad de tiempo en entrenarse. Esto se justifica ya que la red neuronal debe crear una mayor cantidad de bits al codificar una palabra y así mismo descartar una mayor cantidad de bits al decodificar una pa-

[1] https://colab.research.google.com/

labra codificada (en el receptor). Esto se puede apreciar de mejor forma al observar que el tiempo de entrenamiento del experimento 3 (E3) es sustancialmente reducido en comparación con aquel reportado para los demás experimentos sin feedback. Note que la tasa de codificación del experimento 3 es la menor de todas teniendo una tasa de 3/4 y codificando únicamente palabras de 12 bits.

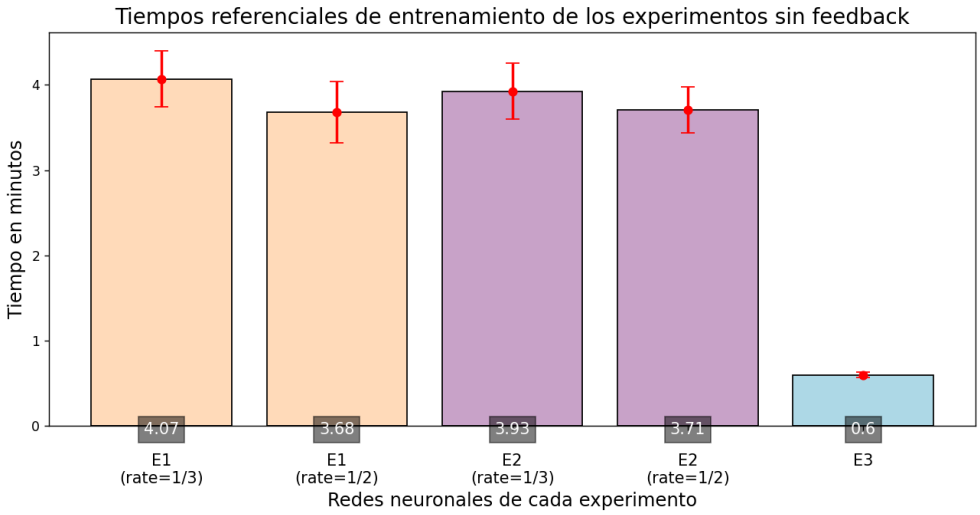

Figura 3.17: Tiempos de entrenamiento de experimentos sin feedback.

Con respecto a los experimentos con feedback, se tiene tiempos de entrenamiento mayores ya que estos usan redes neuronales mas complejas como GRU y LSTM.

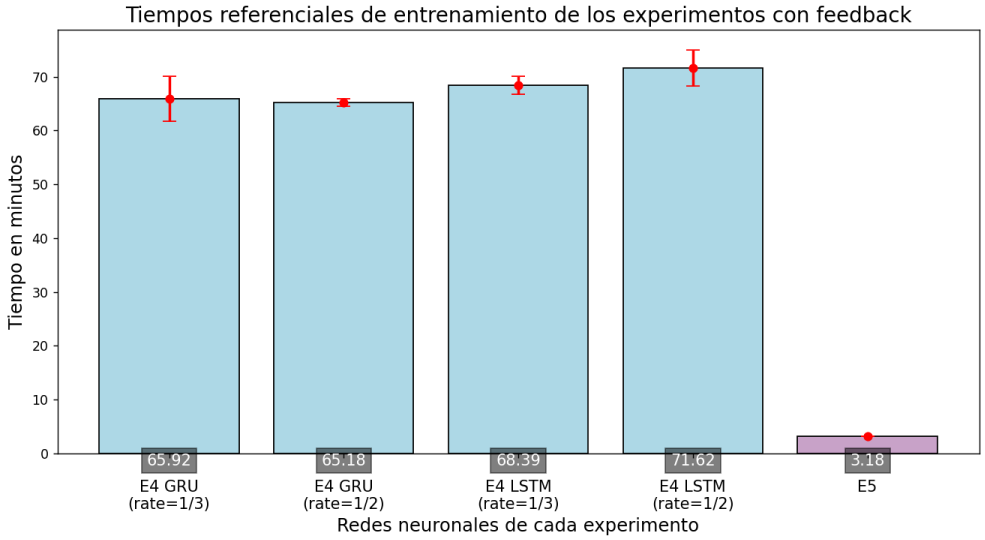

Figura 3.18: Tiempos de entrenamiento de experimentos con feedback.

En la Figura 3.18 se aprecia un ligero cambio en el comportamiento de los tiempos

de simulación. Nótese que los experimentos realizados con redes LSTM tienen un tiempo de duración en promedio mas largo que los realizados con GRU, esto debido a un mayor requerimiento de memoria de las redes LSTM. Adicionalmente se ratifica que la red con el menor tiempo de entrenamiento es aquella con la mayor tasa de codificación (experimento 5 –E5).

Capítulo 4 Experimentación, resultados y análisis

En esta Sección se presentan los resultados de cada una de las implementaciones de códigos convencionales y también cada experimento que se mencionó en la Sección 3, con el fin de obtener conclusiones, observaciones y analizar el desempeño de nuestra propuesta.

Para los experimentos de los modelos ya entrenados, se emplea la metodología explicada en la Sección 3.3.6, en la que se lleva a cabo la prueba de 20 épocas (o iteraciones), en cada época se prueba la transmisión de 100000 mensajes. El resultado de cada época se obtiene 20 valores de Bit Error Rate (BER) por cada probabilidad de error de canal evaluada. Estos valores se promedian y también se calculan sus respectivos intervalos de confianza. El número de épocas se estableció con base en el tiempo de ejecución de cada experimento.

En contraste, para las implementaciones de los códigos convolucionales (a diferentes rates) se utiliza una cantidad reducida de datos en comparación a los modelos neuronales. En particular, la fuente de datos con ese código convencional es de 5000 palabras, debido a la alta complejidad computacional que tiene este programa, el cual limita en tiempo la ejecución de épocas con mayor cantidad de palabras. Para el resto de códigos convencionales también se usan 100000 palabras. En relación al número de épocas, también se realizan 20, de esta manera se promedian y se obtienen los intervalos de confianza.

Luego de tener los resultados, se procede a realizar las gráficas de BER frente a probabilidades de error, esto en función de las tasas de codificación y al tipo de experimento realizado. Algo a destacar es el uso de la herramienta “automeris.io” que se puede encontrar en [64]. Este recurso fue empleado para la extracción precisa de datos de algunas imágenes presentadas en el marco teórico, con el objetivo de generar una comparación de resultados más completa para ciertos códigos convencionales.

4.1. Evaluación de redes sin retroalimentación con diferentes tasas de codificación frente a códigos convencionales

Rate = 1/2

Los primeros modelos a evaluar son los mencionados en los Experimentos 1 y 2 con tasa de codificación 1/2. El resultado de estos modelos se compara con códigos polares, convolucionales y datos enviados sin codificación. El resultado obtenido se muestra en la Figura 4.1.

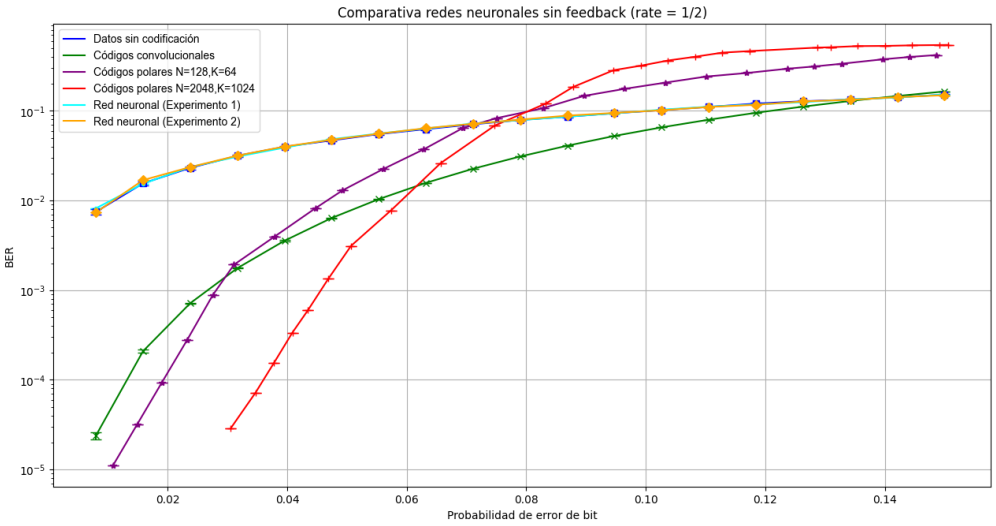

Figura 4.1: Comparativa redes neuronales sin retroalimentación con rate 1/2.

Puede observarse con colores cyan y amarillo, el resultado de los Experimentos 1 y 2 respectivamente. Se puede ver que el uso de redes neuronales en este caso no presenta mayores mejoras, pues en color azul se presenta el resultado de enviar la información sin codificar y podemos apreciar que estas tres curvas tienen un desempeño similar. Además de eso, observamos que los códigos polares y convolucionales tienen un mejor rendimiento. Sin embargo, esto pasa solo hasta una probabilidad de 0.08 para los polares, después de eso, enviar sin codificar resulta mejor, a excepción de los códigos convolucionales, pues estos sí tienen mejor rendimiento casi hasta el final del intervalo de prueba.

En resumen, para una tasa de codificación de 1/2, el uso de redes neuronales no es viable para un BSC con las arquitecturas propuestas, debido a que no se aprecia ninguna mejora a comparación de enviar los datos sin codificar y los códigos convencionales tienen un mejor rendimiento.

Cabe mencionar que a pesar de que los códigos convolucionales exhiben un mejor resultado, el punto negativo se encuentra en la eficiencia de estos para la codificación, pues este tipo de codificadores son los que más tiempo y complejidad para el proceso de codificación demostraron en nuestros experimentos, teniendo que bajar el tamaño de la fuente de datos para que sea viable la simulación.

Rate = 1/3

La siguiente evaluación a realizar es con una tasa de 1/3, en este escenario se tiene algunos códigos convencionales para la comparación, estos son los códigos espinales con y sin perforación, códigos de repetición, los turbo códigos y el resultado de los modelos de redes neuronales entrenados. En la Figura 4.2, se presenta la comparación con los códigos convencionales.

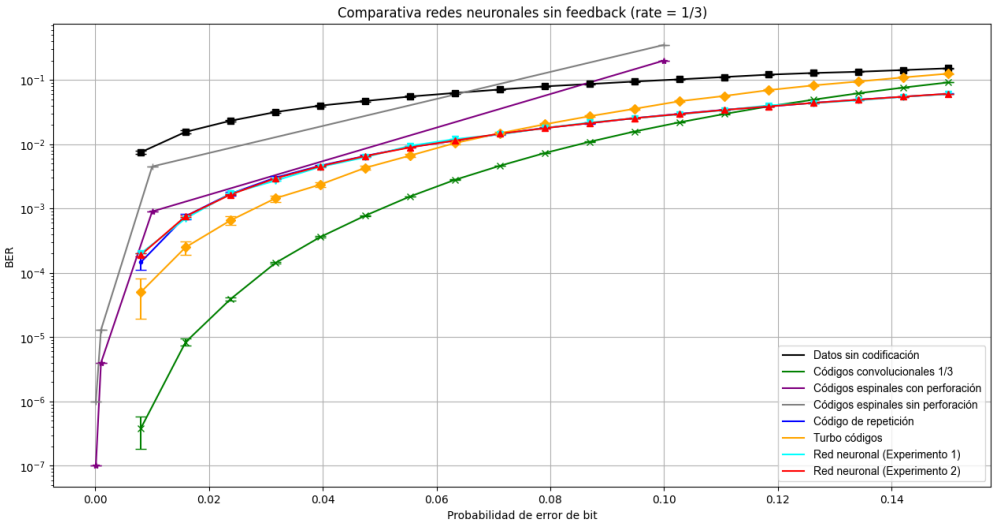

Figura 4.2: Comparativa redes neuronales sin retroalimentación con rate 1/3.

En la Figura 4.2, en color cyan y rojo, se observa la curva de BER de los Experimentos

1 y 2 anteriormente mencionados. En esto se logra distinguir que los resultados de estos dos se encuentran sobrepuestos en la curva de BER del código por repetición de tasa 1/3. Es decir, que se puede interpretar que las redes neuronales tanto para el Experimento 1 y 2 descubrieron un código que tiene un comportamiento similar a un código de repetición en el canal binario simétrico. El análisis sobre el descubrimiento que hizo el codificador se desarrollará en la Sección 4.4.

El resultado de la Figura 4.2 frente las gráficas de los códigos convencionales hacen ver que, para probabilidades de error en el canal, relativamente bajas, el uso de los códigos tradicionales evidentemente es mejor, a pesar de ello, para el caso de los turbo códigos cuando superan la probabilidad de aproximadamente 0.07, tiene un desempeño por debajo que los códigos descubiertos por las redes neuronales y los códigos de repetición. Los que se quedan fuera de competencia son los códigos espinales, pues estos tienen el peor desempeño, tanto así que para una probabilidad mayor a 0.07 en el caso de códigos espinales sin perforación y para mayores de 0.09 para los que poseen perforación, es mejor enviar la información sin codificar.

En este escenario, también se aprecia que el uso de técnicas de aprendizaje automático efectivamente contribuye a mejorar el proceso de comunicación sobre un BSC. Esta mejora que realiza, si bien se asemeja a los resultados de usar un código de repetición, demuestra que supera a los turbo códigos y en una pequeña sección a los convolucionales para probabilidades de error de canal más altas.

Es importante analizar otros factores, uno de ellos la complejidad computacional de cada uno de estos códigos, pues al final el uso de redes neuronales, una vez ya entrenadas, se reduce a simples combinaciones lineales de las entradas a los modelos, mientras que el diseño de un turbo codificador o un codificador convolucional representa un esfuerzo mayor.

Así mismo, a nivel de desempeño en las pruebas realizadas para los modelos basados en redes neuronales se demoran en promedio 8 minutos, considerando una ejecución para 100000 palabras durante 20 épocas. En contraste, el turbo codificador se demora un día entero (aproximadamente 24 horas) mientras que el convolucional un par de días (aproximadamente 48 h), por ende, en este sentido, usar el modelo basado en redes neuronales es mejor. Recuérdese también que las palabras a codificar tienen

una longitud de 50 bits.

A pesar de estas observaciones experimentales, consideramos que se podría realizar un mecanismo híbrido para que los turbo códigos o los convolucionales trabajen mientras que la probabilidad de error del canal esté por debajo de cierto valor para que luego de ese umbral entren en acción los modelos de redes neuronales.

Al analizar con más detenimiento los códigos con repetición se puede concluir que prácticamente los resultados son similares, por lo que la decisión de usar uno u otro dependería mucho de la dificultad de implementación de los códigos. Si bien, en el código repetición la parte decisiva se encuentra en el decodificador que elige el resultado por votación, las redes neuronales son cajas negras que tienen entradas y dentro realizan combinaciones lineales para dar un resultado, entonces, la elección podría quedar a opinión del usuario.

Rate = 3/4

Por último, se evalúa el modelo diseñado para una tasa de codificación de 3/4, para la comparación se consiguió implementar un código de Hamming (11,15), de una tasa de codificación similar, aproximadamente de 3/4. Así mismo, se enviaron los datos sin codificar y el resultado obtenido es el que se presenta en la Figura 4.3.

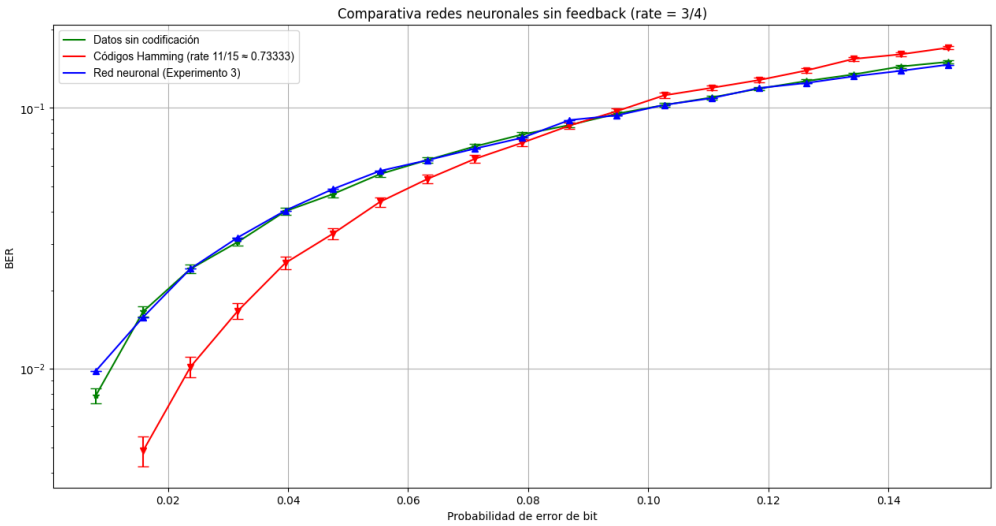

Figura 4.3: Comparativa redes neuronales sin retroalimentación con rate 3/4.

De la Figura 4.3, se tiene un resultado esperado, pues al final aquí la red por sí sola no influye altamente para la protección de errores, pues solo envía 1 bit por cada 3 de información, por ello, en este experimento era esperable que la red neuronal tenga el mismo que resultado que enviar la información sin codificar o peor, sin embargo, en este caso la red no empeoró la situación, pero tampoco la mejoró.

Por otro lado, aquí los códigos de Hamming son evidentemente mejores, pero sigue el mismo patrón que experimentos anteriores, en cierto punto, correspondiente a una probabilidad de error de canal de 0.09, es preferible enviar la información sin codificar. Por ello, quizá la búsqueda de otras arquitecturas de redes neuronales para mejorar esa necesidad que presentan los códigos convencionales en canales binarios simétricos sea importante a considerar.

4.2. Evaluación de redes con retroalimentación con diferentes tasas de codificación frente a códigos convencionales

Esta Sección presenta los resultados de evaluación de los modelos de redes neuronales cuando se dispone de retroalimentación. Claramente, para hacer las evaluaciones se han dividido las redes neuronales por su tasa de codificación. Cabe aclarar que en todos los experimentos realizados con retroalimentación se ha utilizado una probabilidad de error en el canal de feedback de 0.0000001.

Rate = 1/2

En primer lugar, se ha evaluado la red neuronal con retroalimentación para una tasa de 1/2. Esta se ha comparado con diversos códigos convencionales, los cuales se muestran en la Figura 4.4.

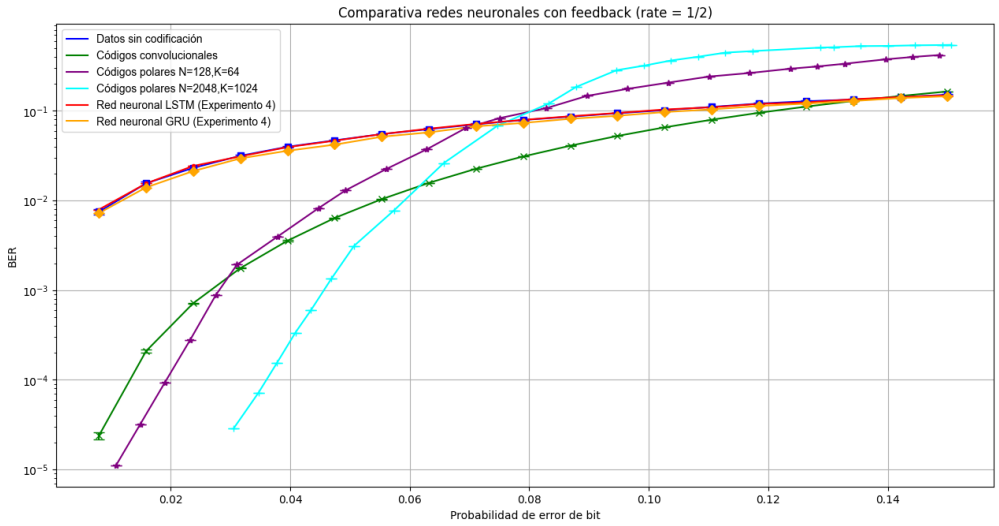

Figura 4.4: Comparativa de redes neuronales con retroalimentación con tasa 1/2.

Observe que el desempeño de las redes neuronales utilizadas en el experimento 4 es muy similar al desempeño de los datos sin codificar. Es decir, en estos casos tanto el codificador como el decodificador no están aprovechando el bit de codificación para mejorar la robustez del código ante el ruido del canal. No obstante, al observar el comportamiento de todos los códigos presentados en la Figura 4.4, podemos ver que para probabilidades superiores a $q = 0.08$ estos códigos se comportan de una mejor manera que los códigos polares. Esto no demuestra la utilidad de los códigos descubiertos en esta Sección, ya que únicamente indica que en esos casos es mejor enviar la información sin ningún tipo de codificación.

Rate = 1/3

En cuanto a la tasa de codificación de 1/3, se puede ver en la Figura 4.5 las distintas curvas de BER tanto para las redes neuronales como para códigos convencionales.

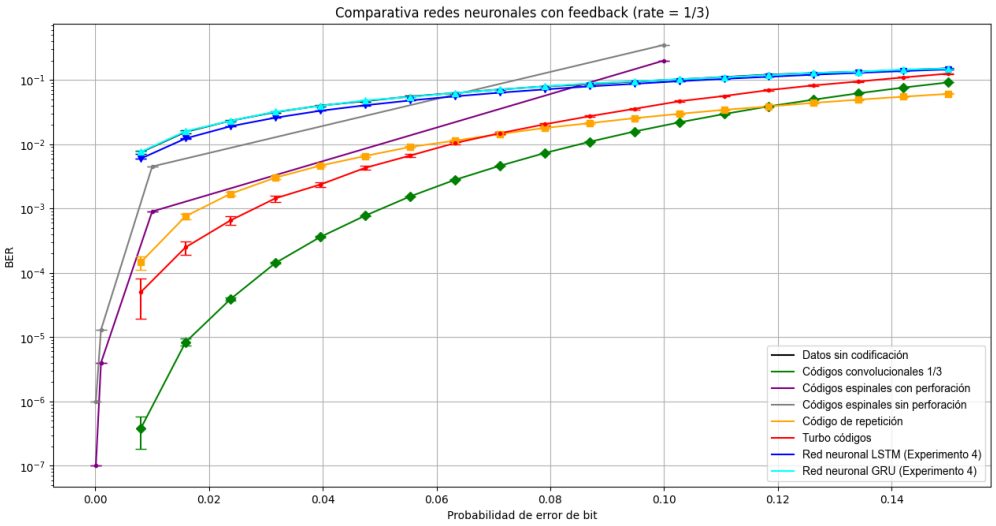

Figura 4.5: Comparativa de redes neuronales con retroalimentación con tasa 1/3.

Podemos observar que los resultados son un poco más alentadores que los hallados en la Figura 4.4 para un rate mayor (1/2). Si miramos la gráfica del experimento realizado con una red Gated Recurrent Unit (GRU), podemos notar que esta red no tiene buenos resultados. Sin embargo, al observar la curva de BER generada por los resultados obtenidos al utilizar una red Long Short-Term Memory (LSTM), podemos notar una ligera mejora en comparación con los datos sin codificar. Esta mejora se extiende a medida que la probabilidad de error baja, presentando aproximadamente el 75 % de los errores que presentan los datos transmitidos sin codificación.

Rate = 3/4

Por último, se puede observar en 4.6 cómo se comparan los códigos generados con una red neuronal ante los códigos Hamming en un canal binario ruidoso.

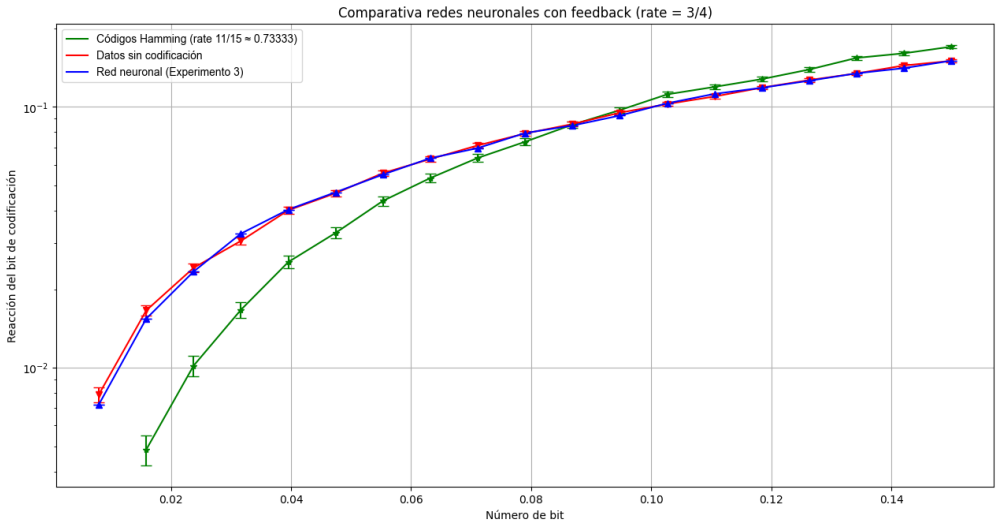

Figura 4.6: Comparativa de redes neuronales con retroalimentación con tasa 3/4.

Podemos observar un resultado similar al obtenido con la red con tasa 1/2. Note que la curva de desempeño del código generado con una red neuronal que usa retroalimentación está sobrepuesta a la curva de los datos enviados sin codificación. Obviamente, la curva de desempeño de los códigos Hamming está muy por debajo de ambas. Esto quiere decir que, para esta arquitectura y esta tasa de codificación, las redes neuronales no son funcionales. Cabe mencionar también que el tiempo utilizado para la obtención de datos de las redes neuronales es significativamente mayor al tiempo tomado por el codificador y decodificador de los códigos Hamming (15,11). Es necesario aclarar que la comparación puede resultar un poco extraña debido a la diferencia de tasas; sin embargo, consideramos que la comparación es relativamente justa ya que la diferencia de tasas es de un valor aproximado de 0.01666.

4.2.1. Análisis sobre el uso del canal de retroalimentación

Con los resultados presentados en esta Sección se considera valioso hacer una comparación entre los códigos descubiertos con el uso de retroalimentación y sin el uso de la misma, esto lo podemos ver de mejor forma en la Figura 4.7.

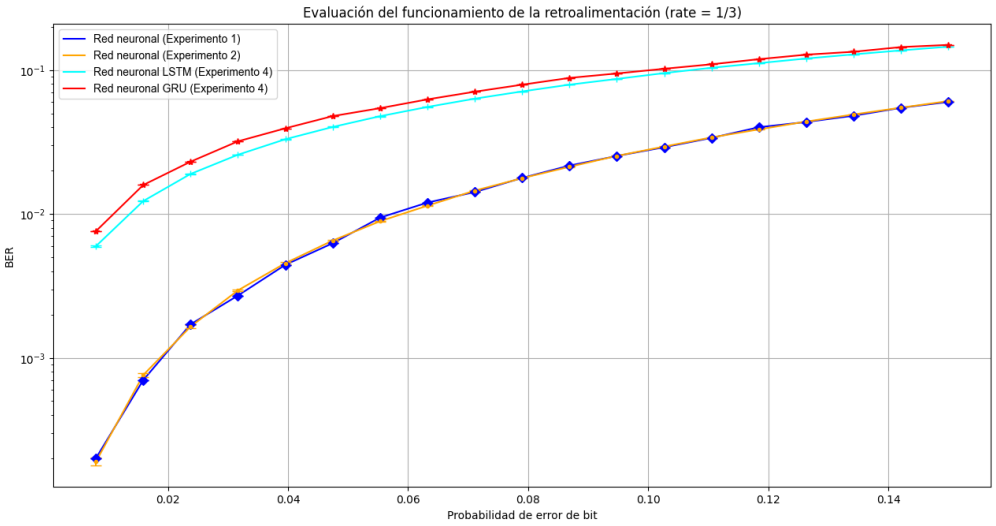

Figura 4.7: Evaluación de la inclusión de retroalimentación (tasa = 1/3).

Claramente puede notarse la diferencia de resultados entre usar retroalimentación ruidosa y no usarla. De manera sorpresiva, el hecho de utilizar retroalimentación genera resultados de menor desempeño que aquellos generados con redes sin retroalimentación. Tal como se puede notar, las redes sin retroalimentación funcionan mucho mejor que las redes con retroalimentación. A pesar que el experimento realizado con una red LSTM presenta una ligera mejora no es suficiente como para que esta supere a las redes sin retroalimentación.

Otro punto importante es el tiempo de ejecución. Las redes neuronales con retroalimentación se toman un tiempo considerablemente mayor tanto en su etapa de entrenamiento como en su etapa de prueba, esto evidentemente se debe a la complejidad que presentan las redes neuronales recurrentes, sin embargo, esta mayor cantidad de tiempo utilizado en el entrenamiento no se vio reflejada en los resultados finales.

El hecho de que la disponibilidad de retroalimentación no contribuya en el proceso de codificación para mejorar los resultados de BER constituye un objeto de investigación futura. En particular, las dificultades inherentes presentadas por un BSC en el proceso de entrenamiento incorporan retos que requieren un proceso de análisis más profundo y que deja abierta las puertas para extender el trabajo presentado en este documento. Entre las dificultades acarreadas por el modelo de canal se puede men-

cionar la necesidad de utilizar funciones de salida dura (como heaviside) para acoplar las señales a las restricciones propias del canal (las entradas solo pueden ser 0 o 1, y no números reales). En nuestra opinión estas funciones de activación imponen dificultades adicionales en los modelos de entrenamiento basados en aprendizaje supervisado puesto que contribuyen a los problemas de desvanecimiento de gradientes durante su proceso su propagación hacia atrás para la corrección y ajuste de pesos.

4.3. Evaluación de los tiempos de procesamiento de codificadores basados en Deep Learning y codificadores convencionales

En esta sección se analiza los tiempos de procesamiento de los codificadores basados en Deep Learning que presentaron un resultado comparable con los codificadores tradicionales, como se presentó en las Secciones 4.1 y 4.2.

Con un enfoque experimental, se mide el tiempo que tarda en codificar y decodificar cada modelo con el objetivo de percibir el esfuerzo computacional en la ejecución completa del proceso de codificación y decodificación. Se considera entonces el procesamiento de un batch de 100 palabras para cada uno de los codificadores, este proceso se repete 100 veces en cada uno de los modelos y codificadores tradicionales que fueron implementados. Estos tiempos se promediarán y se obtiene el respectivo intervalo de confianza para la generación de gráficas comparativas. Así mismo, para que sea una comparación justa, tanto los modelos basados en Deep Learning como los codificadores tradicionales se ejecutan en Google Colab empleando un CPU, asegurando así una igualdad de recursos disponibles para cada escenario.

Con base en los resultados de BER, los experimentos 1 y 2 con un rate de 1/3 y el experimento 4 con rate 1/3 y empleando capas LSTM fueron los que presentaron una mejora frente a el envío de mensajes sin codificar. En función a esto se elige para esta prueba los codificadores convolucionales, turbo códigos y repetición con un rate de 1/3 para llevar a cabo la evaluación de los tiempos de procesamiento. La Figura 4.8, muestra el tiempo que le toma a cada experimento, los procesos de codificar y decodificar un *batch* de 100 palabras.

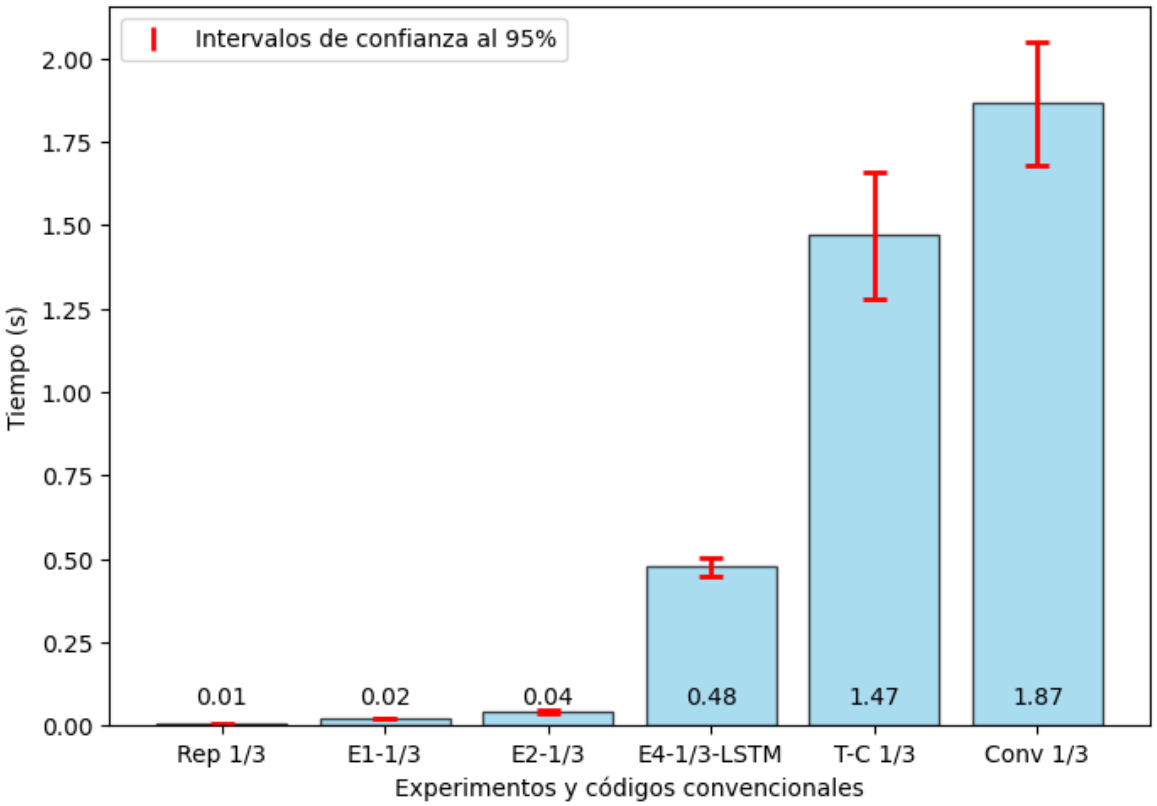

Figura 4.8: Tiempos de procesamiento para codificación y decodificación de un batch de 100 palabras para cada experimento y codificador tradicional.

En la Figura 4.8, se observa que el codificador con el menor tiempo de procesamiento para codificar y decodificar un batch de 100 palabras es el de repetición con rate 1/3, seguido de los 3 experimentos que presentaron resultados (E1, E2 Y E4) y dejando en evidencia que el uso de capas LSTM y obviamente, que la inclusión del canal de retroalimentación en el experimento 4 producen un aumento considerable en el tiempo de procesamiento, que se incrementa de 0.04 segundos en el experimento 2 a 0.48 segundos por lote. Este incremento se traduce como un incremento notable del esfuerzo computacional en la ejecución del experimento. En contraste, la ejecución de codificación y decodificación para Turbo Códigos (T–C) y para códigos convolucionales (Conv) exhiben tiempos de ejecución de tres o hasta cuatro veces el tiempo de procesamiento del experimento 4 para un lote (*batch*) de 100 palabras, llevando a inferir que la complejidad computacional aumenta de manera importante.

Este breve análisis del tiempo de procesamiento provee una percepción tangible de

los niveles de complejidad computacional de cada una de las implementaciones y evidencia que los codificadores basados en *Deep Learning* (una vez entrenados) poseen un mejor desempeño en términos de carga computacional y tiempos de codificación y decodificación. Este hecho se justifica en que los modelos basados en redes neuronales una vez optimizados se reducen a un conjunto de combinaciones lineales que se realizan en función de una entrada y que brindan una salida, lo cual constituye operaciones matemáticas más simples a aquellas comparadas con un codificador convolucional o un codificador basado en Turbo Códigos, los cuales se sustentan en una operación significativamente más intrincada.

4.4. Interpretación del comportamiento de codificadores entrenados

En esta Sección se presenta nuestra interpretación o entendimiento de lo que en general, realiza el codificador resultado de los experimentos y del entrenamiento para aquellos casos en los que se cuenta con resultados positivos o demuestran algún intento por corregir errores de transmisión.

Para el Experimento 1 con tasa de codificación 1/3, en la Figura 4.2, pudimos observar un comportamiento de desempeño en términos del BER parecido al de un codificador por repetición de 1/3. Esto podría sugerir que el codificador descubierto por el proceso de entrenamiento es efectivamente el código de repetición.

Para aclarar esta hipótesis, realizamos un nuevo experimento, donde se envían mensajes que están codificados únicamente con ceros o únicamente con unos. Este enfoque facilitaría la observación de cómo responde el codificador frente a las dos posibilidades de entrada.

Se podría pensar que enviar tramas de información con solo ceros o solo unos podría sesgar de alguna forma los resultados que se obtienen, sin embargo, puesto que la probabilidad de error para cada bit es independiente con respecto a la anterior y puesto que el modelo ya se encuentra entrenado, entonces, no modifica de ninguna manera el escenario de prueba o el comportamiento del modelo.

Tras explicar esto, observe la Figura 4.9, en la que se presenta el comportamiento del codificador, donde la idea es presentar de cada 100 palabras transmitidas, qué

es lo que hace el codificador con cada uno de los bits. Por ejemplo, considere que cada palabra enviada tiene 50 bits; si para el bit 1 transmitido de cada 100 palabras, su correspondiente el bit redundante codificado $c_{1,1}$ resulta “1” en 30 ocasiones y “0” en 70 ocasiones, entonces, se esperaría que en esa posición la tonalidad de color azul sea de un 30 % y la de cero sea un 70 %, lo que indicaría que se enviaron más ceros que unos. Es decir que la intensidad de azul refleja si se enviaron más unos que ceros.

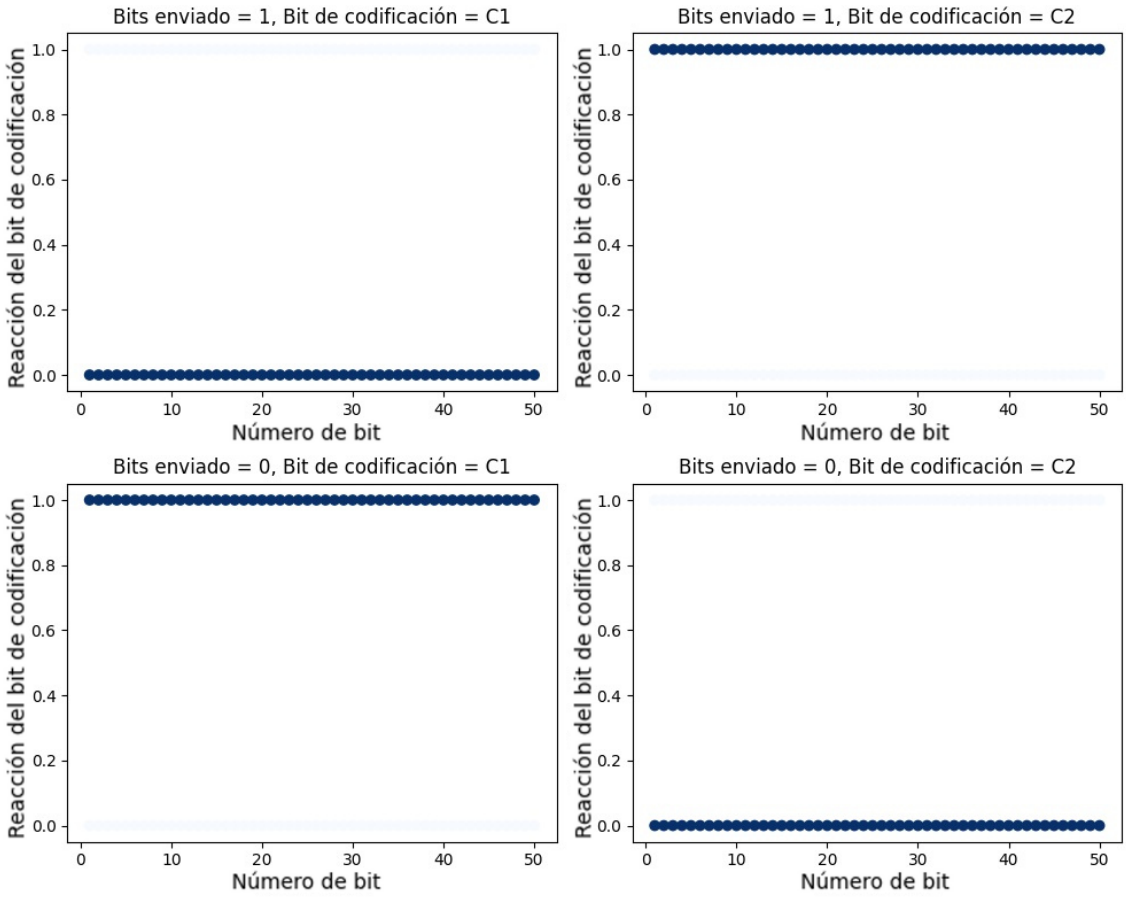

Figura 4.9: Funcionamiento del codificador del Experimento 1, en función de la información de ingreso.

De la Figura 4.9, se puede notar en las imágenes superiores la reacción del codificador al entregar los bits redundantes (de codificación) obtenido por la red neuronal $c_{1,1}$ y $c_{1,2}$ cuando la fuente de información son palabras de 50 bits donde cada bit es “1”. Aquí se puede ver que el codificador establece el bit de codificación $c_{1,1}$ como un “0” en el 100 % de las ocasiones y en “1” el bit de codificación $c_{1,2}$. En otras palabras, descubrió que lo mejor que puede hacer para protegerse de los errores es alternar

ceros y uno en los bits de codificación.

Por otro lado, en las imágenes de abajo en la Figura 4.9 la fuente de información posee palabras de 50 bits, pero donde todos los bits son “0”. Aquí el codificador funciona al revés, por lo que el bit de codificación $c_{1,1}$ siempre envía “1” y en el bit de codificación $c_{1,2}$ envía “0”.

Esto es interesante, porque este resultado sugiere que el codificador de alguna forma etiqueta al bit de información con valor “1” como “101” y al bit con valor de información “0” como “010” y realizando esto, consigue una curva de BER similar a la de un código de repetición.

Este proceso que realiza tiene mucho sentido, pues, para que un “1”, que codificado corresponde a “101”, cambie a “0” debe cambiar en sus tres bits para convertirse en “010”. Ahora analizando de manera un poco más detalladamente, el código “010” también puede cambiar a “101”, cambiando sus tres bits, sin embargo, observando cuantos errores es capaz de soportar sin que se confunda el “0” o el “1”, nos damos cuenta que solo puede resistir el error, de un solo bit de los 3. Esto se puede ver en la Tabla 4.1.

Tabla 4.1: Posibilidades de cambio de palabras codificadas si hay un error en los símbolos resultantes del experimento 1.

Si 101 cambia en un solo bit	**Si 010 cambia en un solo bit**
111	110
100	000
001	011

Como se puede ver en la Tabla 4.1, esas son todas las posibilidades existentes si se presenta un error en un solo bit para cada palabra. Observamos que, de alguna manera, si existe un solo error, el decodificador podrá identificar si se trata de un “1” o un “0”.

Lamentablemente, si existen dos errores, estas posibles combinaciones se mezclan, por ejemplo, si “101”, cambia a “110”, posiblemente de decodificaría un “0” y por ende un error.

Esto sugiere una explicación plausible de por qué la curva de BER es similar al có-

digo de repetición, y es por qué al igual que en repetición, solo se puede resolver un error, es decir, tienen la misma capacidad para solventar las pérdidas de información. Además de esto, es interesante ver cómo el codificador asigna un símbolo a un bit con valor de “1” y otro símbolo un bit con valor de “0”, de tal forma la distancia de Hamming entre las 2 palabras sea la mayor en una familia de palabras de 3 bits.

El mismo análisis se lleva a cabo para el codificador del experimento 2 con tasa de codificación 1/3. En la Figura 4.10 se presenta el comportamiento del modelo bajo el mismo escenario de prueba que el modelo del Experimento 1.

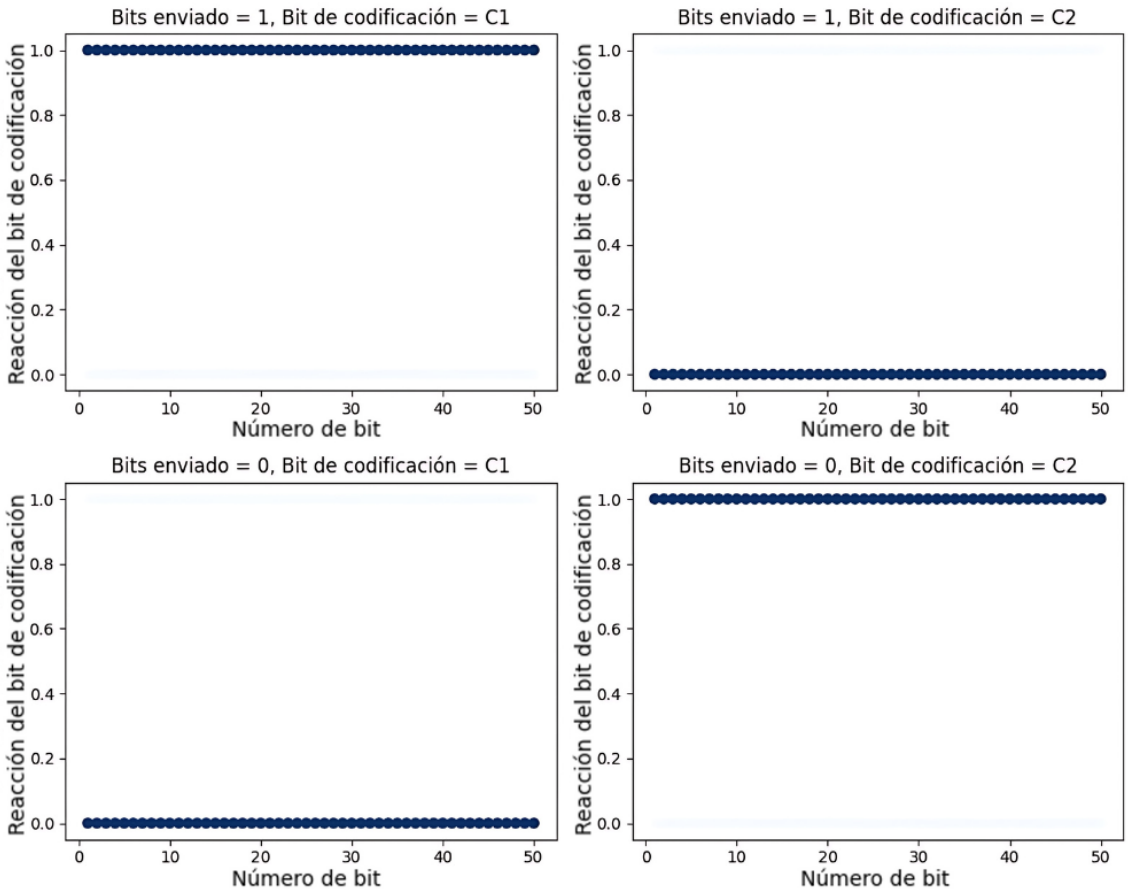

Figura 4.10: Funcionamiento del codificador del experimento 2, en función de la información de ingreso.

En la Figura 4.10, observamos que ahora el modelo asigna al bit con valor de “1” como el código “110” y al bit con valor de “0” se le asigna “001”. La distancia de Hamming entre estas 2 palabras es 3, es decir, la máxima en la familia de palabras de 3 bits. Esto logra un mismo resultado que la red del Experimento 1, pues si ahora seguimos la Tabla 4.2, observamos que el modelo es capaz de resolver un error por

bit transmitido.

Tabla 4.2: Posibilidades de cambio de palabras codificadas si hay un error en los símbolos resultantes del Experimento 2.

Si 110 cambia en un solo bit	**Si 001 cambia en un solo bit**
111	110
100	000
001	011

Observando que para esta red se encuentra un comportamiento similar al descrito arriba, notamos que, de hecho, los símbolos que se pueden asignar al bit de valor “0” o al bit con valor “1” en una familia de palabras de 3 bits, deben cumplir con la condición de tener la distancia de Hamming más alta de la familia de palabras de tres bits, es decir, la distancia debe ser 3. Con este hecho, aseguramos un código capaz de resolver un error por cada símbolo de tres bits enviados.

Al igual que el Experimento 1, este experimento posee una curva de BER similar a un código de repetición, debido a que tiene la misma capacidad para solventar errores, sin embargo, la metodología que aplica para lograr esto es diferente, pues para las redes neuronales no siempre se repiten los unos o los ceros, sino que se asigna un par de símbolos con distancia de Hamming 3, debido a que es la distancia más grande en una familia de palabras de 3 bits.

De hecho, podríamos indicar que de cierta manera para la tasa de codificación de 1/3, el código de repetición es un caso puntual del código descubierto por los modelos, pues la distancia de Hamming entre “111” y “000”, también es 3, entonces, son 2 posibles símbolos que se pueden asignar.

Por último, se analiza el código descubierto en el Experimento 4, donde se usa un canal de retroalimentación, por lo que este sería un código de canales binarios simétricos con retroalimentación. El resultado obtenido del trabajo que realiza la red, con la misma metodología realizada para los Experimentos 1 y 2, se presenta en la Figura 4.11.

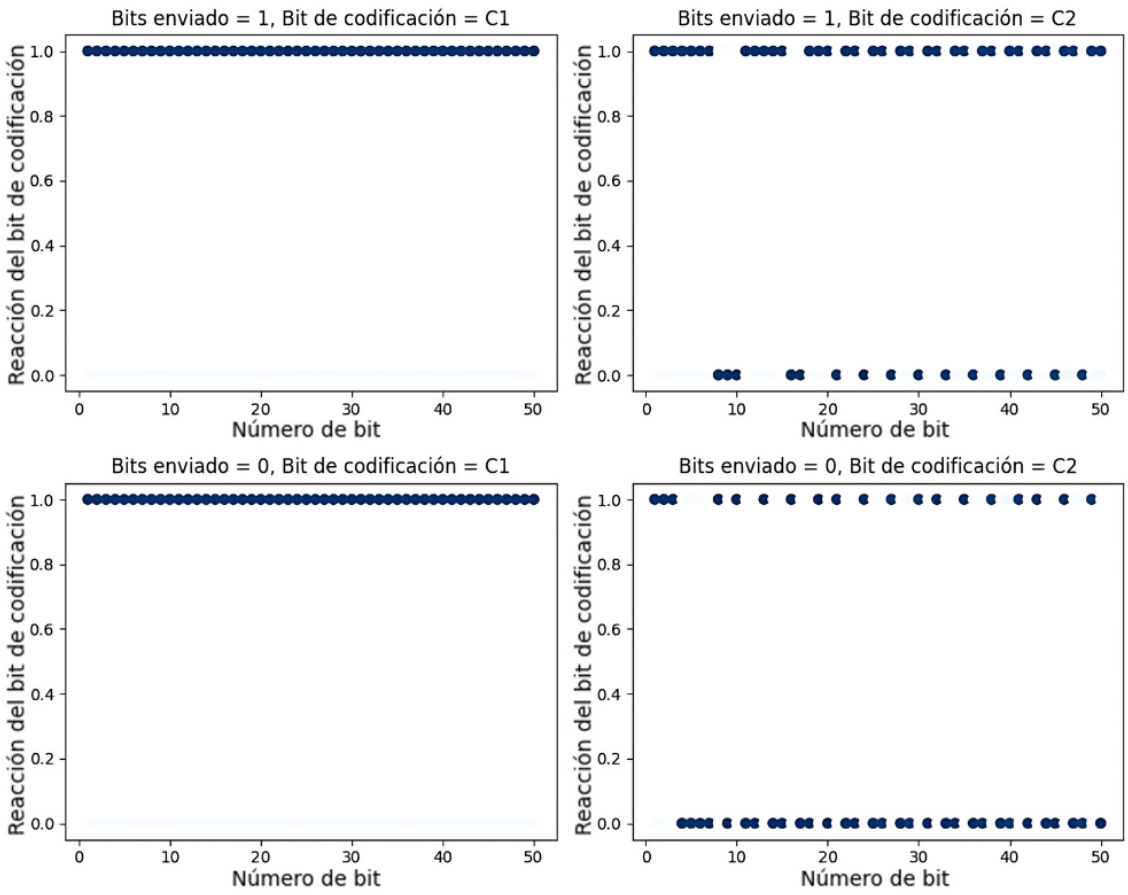

Figura 4.11: Funcionamiento del codificador del experimento 4 con redes LSTM , en función de la información de ingreso.

Tal como se presentó en la Subsección 4.2, al usar el escenario con retroalimentación e implementando capas tipo LSTM , se obtuvo una pequeña mejora frente a enviar los datos sin codificar. Con base en este hecho, en la Figura 4.11, se puede observar que independientemente si el bit enviado es un “0” o un “1”, el bit de codificación $c_{1,1}$, siempre se mantiene en 1. Lo interesante está en el bit de codificación $c_{1,2}$, pues al realizar varias pruebas, se ha repetido el mismo patrón. De igual manera, si la palabra contiene solo ceros, el patrón también es el mismo. Es decir, de alguna manera, información que llega de la retroalimentación ayuda a mejorar un poco en el proceso de codificación, sobre todo en probabilidades de error de canal bajas.

Claramente, el código que se descubrió cuando se dispone de feedback no es muy bueno, sin embargo, hay que considerar que en un canal binario simétrico la cantidad de información que se pierde es demasiada ya que solo se tiene dos estados (cero y uno). Esto de alguna manera puede hacer que a pesar que la información de re-

troalimentación llegue por un canal incluso perfecto, no sea de mucha ayuda durante el entrenamiento de un modelo neuronal. Nótese que al usar algoritmos basados en retropropagación del gradiente, dada la naturaleza binaria del canal, el gradiente no brinda suficiente información para poder realizar un mejor trabajo. A pesar de ello el hecho de que exista alguna mejora, lleva a pesar que implementando otras arquitecturas o utilizando diferentes métodos de entrenamiento los resultados puedan mejorar.

De la Figura 4.11, la mejor interpretación que podemos dar es que la retroalimentación intenta ayudar a la modificación del bit de codificación $c_{1,2}$, y que de hecho, contribuye para que el BER se reduzca ligeramente en comparación con el envío de datos sin codificación. Una generalidad es que, al enviar bits con valor de “1” como información, la mayoría de veces el bit de codificación $c_{1,2}$ es “1”, y lo mismo pasa si la información responde a un bit de valor “0”, con la diferencia que ese bit de codificación en mayor proporción es cero. Esto podría llevar a encontrar algún patrón más complejo, sin embargo, consideramos que se requiere un mayor análisis sobre nuestra interpretación del comportamiento del codificador del Experimento 4 con redes LSTM.

4.5. Limitaciones y complicaciones en el contexto de un BSC.

La principal limitación encontrada en el desarrollo de este trabajo y al implementar la red neuronal utilizada en esta tesis ha sido la dificultad de optimizarla para el modelo de canal objeto del estudio: Binary Symmetric Channel (BSC). El problema radica en la cantidad de información que se entrega a la red neuronal para su aprendizaje considerando el proceso retropropagación. Al utilizar un canal binario simétrico, se tienen únicamente valores binarios tanto a la entrada como a la salida, por lo que la función de activación utilizada, a la entrada del canal debe ser obligatoriamente un escalón unitario que genere únicamente bits, es decir, valores de 0 o 1. En este punto, surge la dificultad de que la función de escalón unitario no tiene una derivada definida en la librería utilizada, y esta derivada es imprescindible para la retropropagación de la red neuronal. Como solución, optamos por mantener funciones de activación no lineales, como la función sigmoide, seguida de la función “heaviside” que proporcionaba los valores binarios necesarios para nuestro propósito.

Sin embargo, esta no fue una solución definitiva, ya que, al tener únicamente valores binarios a la salida de las redes neuronales utilizadas tanto para el codificador como para el decodificador, toda la información recibida por la red para su entrenamiento tenía una forma binaria lo que dificulta un ajuste más preciso de las conexiones sinápticas de los elementos de procesamiento neuronales de la red. Es decir, este hecho complica la determinación de qué tanto fallaba la red o cuánto acertaba durante el aprendizaje supervisado, y más bien, se orienta hacia determinar solamente si su comportamiento era correcto o no. Esto evidentemente complica el proceso de optimización de la red, ya que no se le entregaba suficiente información. Con la red configurada de esa forma, el entrenamiento generaba resultados desalentadores, por lo que se tomaron ciertas decisiones para facilitar el entrenamiento y mejorar los resultados de la red.

La primera medida fue eliminar la función “heaviside” a la salida del decodificador. Esto se hizo porque no presentaba ningún conflicto con el hecho de tener un canal binario, ya que lo único necesario para mantener la binariedad de la información que ingresa al canal es la función “heaviside” colocada a la salida del codificador. Una vez hecho esto, la cantidad de información entregada a la red para su entrenamiento aumentó considerablemente, lo cual se reflejó en los resultados de BER. Al eliminar la segunda función “heaviside”, la salida de la red permitía que la herramienta usada para la medición del error pudiera precisar mejor la cantidad de error encontrado. Por supuesto, esto se hace únicamente para el cómputo de la función de pérdida, puesto que las palabras decodificadas finalmente, sí experimentan una binarización basada en una función heaviside.

Otra de las decisiones acertadas en el proceso de construcción del codificador y decodificador es la de convertir los bits de cada mensaje a transmitir en una señalización antipodal. Similar a una modulación Binary Phase Shift Keying (BPSK). Esta modulación no genera constelaciones con valores en fase ni cuadratura, sino que únicamente presenta los bits de manera que tengan valores opuestos en términos de presentación. Aunque este pueda parecer un cambio sutil, es sumamente útil al utilizar la red, ya que los parámetros y valores de las neuronas y sus conexiones no serán multiplicados por valores de cero (en caso de tener ese valor en la cadena de bits enviada), sino

que serán multiplicados por valores de "-1". Esto permite que la red actúe de mejor forma, influyendo así de manera más efectiva en la información para su codificación o decodificación. Después de tomar estas dos decisiones los resultados obtenidos por parte de la red mejoraron de forma considerable, dando así paso a la experimentación.

Otra limitación que se presentó al entrenar la red neuronal fue el tiempo de entrenamiento puesto que estos procesos suelen ser muy exigentes a nivel de hardware, lo que resultó en tiempos bastante extensos de entrenamiento y prueba del autoencoder. Afortunadamente, PyTorch cuenta con Compute Unified Device Architecture (CUDA) implementado, lo cual permitió realizar las simulaciones usando una Graphics Processing Unit (GPU) en lugar de una Central Processing Unit (CPU).

Finalmente, una limitación que de igual forma generó problemas fue la implementación de códigos convencionales. La complejidad que conlleva implementar algunos de los códigos utilizados hizo atractiva la idea de buscar librerías o códigos ya implementados. No obstante la mayoría de códigos no están implementados para canales BSC lo que conllevó problemas para adaptarlos al modelo seleccionado en este trabajo. El problema principal que se tenía era que al momento de utilizar los códigos en un canal BSC, no se obtenían buenos resultados en principio. Este problema fue solucionado al modular todos los bits en BPSK, lo que contribuyó a que los codificadores y decodificadores tengan un mayor efecto sobre los bits, es decir, nuevamente partir de una señalización antipodal.

Capítulo 5 Conclusiones, recomendaciones y Líneas Futuras de Investigación

En este trabajo de integración curricular, se han evaluado diferentes enfoques y técnicas de Deep Learning para llevar a cabo la codificación del canal, específicamente sobre un BSC. En particular, se usó Deep Neural Network (DNN), Long Short-Term Memory (LSTM) y Gated Recurrent Unit (GRU). El uso de estas técnicas se puede evidenciar en diferentes escenarios experimentales. A partir de los resultados de nuestros experimentos se determina que la técnica que mejores resultados produjo fue el uso de DNN. Esta técnica consiguió que los modelos logren aprender un código que presenta un desempeño similar a los códigos de repetición a una tasa (rate) específica en términos del Bit Error Rate (BER).

En el caso del uso de redes LSTM, en un escenario con retroalimentación consiguió una ligera mejoría frente al envío de datos sin codificar, sin embargo; dicha mejoría no fue significativa e incluso la interpretación del código descubierto requiere de continuar esta investigación. En relación al uso de redes GRU, los resultados no reflejan una mejoría sustancial a comparación de métodos de codificación tradicionales o frente a otros modelos empleados en este mismo trabajo, particularmente se tenía la hipótesis de que el uso de redes GRU genere mejores resultados ya que en presencia de retroalimentación (feedback) lo esperable es que la función de memoria GRU pueda mejorar el desempeño del modelo, tal como se evidenció en trabajos relacionados como [7] y [8] para canales AWGN. Es importante mencionar que un canal AWGN, si bien se construye con base en restricciones de potencia de transmisión, tiene un modelo matemático de canal que admite números reales, y con ello, que facilita el proceso de entrenamiento y de retro-propagación y descenso del gradiente usado en modelos neuronales.

Los resultados experimentales de este trabajo ratifican las complicaciones y desafíos que representa el uso de un Binary Symmetric Channel (BSC). El hecho de que la información que ingresa y que sale del canal sea binaria, involucra una alteración en la información resultante de los codificadores hacia adelante y de retroalimentación (que corresponde a un número real que se convierte en binario) que se transmite por el canal. A partir de esto, nuestros resultados sugieren que para canales de una sola vía binarios, la mejor técnica a utilizar para implementar un modelo de codificación de

canal son las DNN (redes neuronales profundas totalmente conectadas).

A continuación, respondemos brevemente a cada uno de los objetivos específicos planteados en este proyecto, puntualizando cómo se consiguieron cubrir de manera exitosa, y las conclusiones que se derivan de cada uno:

En respuesta a nuestro primer objetivo específico, cabe mencionar que se diseñaron, implementaron, y evaluaron cinco experimentos basados en redes neuronales de aprendizaje profundo, cuya descripción se encuentra el Capítulo 3. El diseño de estos experimentos ha requerido una revisión exhaustiva del estado del arte, en el contexto del uso de técnicas de Deep Learning aplicadas a la codificación de canal. Este estudio comprende una etapa de aprendizaje a nivel conceptual, en primer lugar, de los principios fundamentales del aprendizaje automático, y seguidamente de su aplicación en nuestra área de interés.

Para la generación de estos espacios de experimentación se consideró una categorización en dos grupos amplios, estos son redes con retroalimentación y redes sin retroalimentación. En los experimentos generados se han utilizado distintas arquitecturas, escenarios y tasas de codificación. Después de observar y analizar los resultados de los experimentos, podemos llegar a la conclusión de que para el modelo de canal seleccionado (BSC) y tomando en cuenta las redes neuronales utilizadas, la mejor opción es emplear arquitecturas sin retroalimentación, técnicas de redes neuronales profundas y tasa de codificación 1/3 para la creación tanto del codificador como del decodificador.

Independientemente de si el enfoque del codificador es generar dos bits redundantes en adición a cada bit de información o, si genera los tres bits que serán transmitidos, esta es la opción que mejores resultados brindó. Además, tal como se mencionó se han obtenido resultados ligeramente positivos con el uso de redes LSTM en escenarios con retroalimentación.

Con respecto al segundo objetivo específico se debe resaltar que cada uno de los experimentos planteados se considera un sistema de aprendizaje profundo para el descubrimiento de códigos. En cada experimento se planteó diferentes ideas que se plasmaron en distintas arquitecturas y métodos que se orientan a buscar el mayor

provecho de las técnicas de Deep Learning y con el objetivo de tener resultados significativos. En este proceso la implementación no solamente se enfocó en uno sino en varios escenarios en los que diversos parámetros deben ser considerados para conseguir el objetivo propuesto. Uno de los parámetros esenciales a modificar en los diversos experimentos es la tasa de codificación (rate) con el fin de poder ubicar los códigos descubiertos por nuestros modelos en una posición comparable con otros códigos convencionales. En cuanto a cambios estructurales que se presentaron en los escenarios planteados se puede recalcar que el más importante fue el uso o no de un canal con retroalimentación. Dado el modelo de canal escogido para este trabajo (BCS) también se puede concluir que el uso de canales sin retroalimentación y empleando redes neuronales profundas logra efectivamente descubrir códigos de desempeño comparable con otros códigos, tal como se indicó en el Capítulo 4 . Los resultados arrojados con el enfoque planteado en este proyecto en el contexto del uso de aprendizaje profundo mostraron que el codificador aprende códigos incipientes, puesto que se descubrió que el codificador genera palabras bajo la premisa de que la distancia de Hamming entre ellas sea la mayor posible en esa familia de palabras. Sin embargo, no realiza acciones más complejas o profundas que involucre la correlación entre bits de un mismo mensaje, como por ejemplo si lo hacen algunos códigos convencionales, como turbo códigos, o convolucionales. Si bien esta es una interpretación válida, se requiere de mayor experimentación para tener resultados concluyentes en el caso de que se incluya un canal de retroalimentación.

Finalmente, en respuesta al tercer objetivo específico, como se evidencia en el Capítulo 4, es imprescindible establecer una métrica comparable que permita contrastar el desempeño de los códigos descubiertos con el enfoque planteado en este proyecto con respecto a códigos convencionales. Al observar los resultados, la conclusión a la que se puede llegar es que generalmente el desempeño de los códigos convencionales es superior a la de los códigos descubiertos con redes neuronales en términos del BER. También es importante notar cómo para probabilidades de error del canal relativamente altas los códigos descubiertos funcionan mejor que los códigos convencionales con un desempeño similar a los códigos de repetición.

El desarrollo de este trabajo, nos ha ayudado a comprender a profundidad el mode-

lo de un BSC, y a pesar que matemáticamente es un canal simple, la funcionalidad intrínseca de este provoca que emplearlo represente un reto en el contexto de la optimización simultánea de un codificador y decodificador basado en técnicas de aprendizaje profundo. Esto se debe a que un BSC suprime una gran cantidad de información generada en el codificador al momento de generar las señales que pueden ingresar a dicho canal. Por ejemplo, en el caso de escenarios que poseen retroalimentación es vital contar con un mecanismo que permita propagar el error hacía el codificador en términos de números reales, de tal modo que el codificador entienda el estado del medio por el que va a transmitir el mensaje y consiga realizar un ajuste. Sin embargo, al ser un canal que modifica directamente la información de cero a uno o viceversa, no se cuenta con la facilidad existente en otros canales como uno caracterizado por Additive White Gaussian Noise (AWGN). En esos canales un canal de feedback puede transportar información más precisa sobre cuánto afectó el ruido del canal a cada transmisión, puesto que no se requiere de funciones de activación de salida dura para acondicionar las señales generadas por los modelos neuronales para poder ser introducidas a los canales. Por ello, se indica que el uso en general de canales binarios establece un desafío completamente distinto que debe ser analizado y con base en ello, establecer diferentes escenarios o arquitecturas para el entrenamiento del modelo. Debe recordarse que el mayor reto que existe al emplear técnicas de Deep Learning para codificación de canal es armar un buen escenario de entrenamiento del modelo. El escenario debe ser armado en base a conocimientos sobre el canal que se usa y sus limitaciones.

5.1. Aportes

En este trabajo se ha generado una contribución importante en relación al uso de técnicas de Deep Learning para el descubrimiento de códigos en canales BSC. Gracias nuestro enfoque experimental se ha podido evidenciar, entre otras cosas, las limitaciones encontradas en la implementar las redes neuronales enfocadas para la codificación de canal. La identificación de estas limitaciones es de suma importancia para futuras investigaciones que se realicen sobre este canal y el uso de este tipo de herramientas. De esta manera, se tendrá un contexto más robusto sobre los desafíos

que representa un BSC y se podrá tomar mejores decisiones en cuanto al diseño de escenarios para el entrenamiento de los modelos. Claramente, durante el desarrollo del proyecto y en este documento se ha buscado no solamente mostrar los resultados de evaluación de los códigos descubiertos por los modelos entrenados sino encontrar una interpretación que se ajuste a los conocimientos enmarcados en la Teoría de la Codificación.

Adicionalmente, la experiencia de desarrollar este trabajo, nos motivó constantemente a buscar diferentes metodologías para la implementación de códigos convencionales, pues la literatura en el contexto de su aplicación en canales BSC, es mas bien escasa. Consideramos que así mismo, constituye un aporte importante la implementación presentada de códigos convencionales para este canal en particular dado que esto facilitará el desarrollo de nuevos proyectos que sigan en la misma línea de investigación experimental.

Puesto que el trabajo realizado posee una cantidad de información realmente útil para otras investigaciones, se habilita un repositorio en Git Hub en [65] con todo el desarrollo ejecutado en este trabajo, de esta manera, todas las implementaciones de códigos convencionales o los modelos de redes neuronales quedan a disposición de cualquier usuario que desee emplear como base este trabajo para iniciar o comparar sus investigaciones en canales BSC.

5.2. Trabajos futuros

Como se mencionó en la Sección 4.5, una de las mayores limitaciones que tiene el canal BSC es que toda la información que entra y sale del canal debe ser binaria. Esta condición obliga el uso de la función “heaviside” para la salida del codificador (sea para el canal hacia adelante o para retroalimentación), la cual causa dificultades para el proceso de retropropagación cuando se usa el método de descenso del gradiente. Esto nos sugiere plantear la hipótesis de que el optimizador basado en el descenso del gradiente no logra realizar el mejor de sus trabajos, dado el desafío que representa el BSC. El desarrollo de este trabajo motivó la búsqueda de herramientas que permitan evaluar dicha hipótesis, que, aunque no corresponde al objetivo central

del trabajo de titulación, puede plantear futuras oportunidades de exploración y mejoramiento de los resultados observados en este documento. En particular creemos que debe explorarse el uso de la librería llamada “Nevergrad”, cuya documentación se puede encontrar en [66], y corresponde a una librería de Python desarrollada por Facebook AI Research (FAIR) y que sirve como plataforma de optimización versátil y eficiente que usa un enfoque diferente el descenso del gradiente. Su objetivo principal es llevar a cabo procesos de optimización sin la necesidad del uso del gradiente, para ello, Nevergrad posee una gran cantidad de algoritmos de optimización que se encuentran libres de gradiente. Algunos de los métodos disponibles de optimización sin gradiente son algoritmo genético, optimización de enjambre de partículas, recocido simulado, adaptación de la matriz de covarianza, entre otros. Si bien esta librería aún se encuentra en desarrollo, se espera que pueda ofrecer mejores resultados en contextos como el planteado en este trabajo.

Así mismo, creemos que puede continuarse el diseño de nuevos y diferentes escenarios y arquitecturas para el entrenamiento de las redes neuronales en canales binarios. Dadas las limitaciones que se señalaron en este documento, se tiene un punto de partida más claro para el planteamiento de nuevas estrategias de aprendizaje que mejore los resultados obtenidos aquí. Para ello, se pueden tomar diferentes metodologías para el entrenamiento de las redes, como el entrenamiento no simultáneo del codificador y decodificador o el uso de heurísticas como la aplicación de entrelazadores que por lo general mejoran el rendimiento para la transmisión de información por el canal.

Finalmente, es también importante considerar otros tipos de canales. En particular éste y muchos de los trabajos revisados en el estado del arte demuestran que el uso de redes neuronales en cualquier área ha representado una evolución importante, por lo que el área de las comunicaciones no se queda atrás. Sin duda, la capacidad para adaptar estas técnicas en distintos escenarios de codificación es lo más complejo puesto que existen canales altamente desafiantes, para los cuales seguramente se debe considerar un sin número de heurísticas que faciliten el entrenamiento de los modelos. Es por ello, que el área de investigación en este tema debe crecer y de esta forma generar alternativas que ayuden a resolver el constante aumento en la demanda

de velocidades de transmisión de información que requieren de canales con técnicas de codificación más confiables.

Referencias

[1] W. Xiong y D. Matolak, “Performance of Hamming codes in systems employing different code symbol energies,” in *IEEE Wireless Communications and Networking Conference, 2005*, vol. 2, 2005, pp. 1055–1058 Vol. 2.

[2] M. Bersali, H. Ait-Saadi, y M. Bensebti, “Performance analysis of polar codes vs turbo codes over AWGN channel,” in *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, 2017, pp. 1–6.

[3] A. A. Rodríguez, F. P. González, J. C. Sueiro, R. L. Valcarce, C. M. Nartallo, y F. P. Cruz, *Comunicaciones digitales*. Pearson, 2007.

[4] A. Li, S. Wu, Y. Wang, J. Jiao, y Q. Zhang, “Spinal Codes over BSC: Error Probability Analysis and the Puncturing Design,” in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020, pp. 1–5.

[5] N. Kaur y A. P. S. Kalsi, “Implementation of polar codes over AWGN and binary symmetric channel,” *Indian Journal of Science and Technology*, 2016.

[6] P. Isasi Vinuela, I. M. Galván León *y otros*, “Redes de neuronas artificiales: Un enfoque práctico,” 2004.

[7] H. Kim, Y. Jiang, S. Kannan, S. Oh, y P. Viswanath, “Deepcode: Feedback Codes via Deep Learning,” *IEEE Journal on Selected Areas in Information Theory*, vol. 1, num. 1, pp. 194–206, 2020.

[8] A. R. Safavi, A. G. Perotti, B. M. Popovic, M. B. Mashhadi, y D. Gunduz, “Deep Extended Feedback Codes,” 2021.

[9] T. Google, “Pytorch, Keras, Tensorflow, Scipy - Explorar - Google Trends,” https://trends.google.es/trends/explore?cat=174&q=%2Fg%2F11gd3905v1,%2Fg%2F11c1r2rvnp,%2Fg%2F11bwp1s2k3,%2Fm%2F01n25r&hl=es, Junio 2024, (Accessed on 06/23/2024).

[10] J. Sensio, “SensIO,” https://juansensio.com/blog/027_pytorch_intro, Agosto 2020, (Accessed on 06/04/2024).

[11] E. Stevens, L. Antiga, y T. Viehmann, *Deep Learning with PyTorch*. Manning Publications, 2020.

[12] C. E. Shannon, “A Mathematical Theory of Communication,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, num. 1, p. 3–55, jan 2001. [En línea]. Disponible: https://doi.org/10.1145/584091.584093

[13] S. Lin y D. J. Costello Jr, in *Error control coding*. Library of Congress, 1983.

[14] C. Berrou, A. Glavieux, y P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1,” in *Proceedings of ICC ’93 - IEEE International Conference on Communications*, vol. 2, 1993, pp. 1064–1070 vol.2.

[15] R. Gallager, “Low-density parity-check codes,” *IRE Transactions on information theory*, vol. 8, num. 1, pp. 21–28, 1962.

[16] A. Viterbi, “Convolutional Codes and Their Performance in Communication Systems,” *IEEE Transactions on Communication Technology*, vol. 19, num. 5, pp. 751–772, 1971.

[17] R. W. Hamming, “Error detecting and error correcting codes,” *The Bell system technical journal*, vol. 29, num. 2, pp. 147–160, 1950.

[18] E. Arikan, “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels,” *IEEE Transactions on Information Theory*, vol. 55, num. 7, pp. 3051–3073, 2009.

[19] H. Simon, “Sistemas de Comunicación,” *Madrid, España. Segunda edición, Limusa Wiley*, 2001.

[20] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[21] B. Tahir, S. Schwarz, y M. Rupp, “BER comparison between Convolutional, Turbo, LDPC, and Polar codes,” in *2017 24th International Conference on Telecommunications (ICT)*, 2017, pp. 1–7.

[22] M. Bersali, H. Ait-Saadi, y M. Bensebti, “Performance analysis of polar codes vs turbo codes over awgn channel,” in *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, 2017, pp. 1–6.

[23] M. Jordan y R. Nichols, “The Effects of Channel Characteristics on Turbo Code Performance,” in *Proceedings of MILCOM ’96 IEEE Military Communications Conference*, vol. 1, 1996, pp. 17–21 vol.1.

[24] J. W. Lee, R. L. Urbanke, y R. E. Blahut, “Turbo Codes in Binary Erasure Channel,” *IEEE Transactions on Information Theory*, vol. 54, num. 4, pp. 1765–1773, 2008.

[25] E. C. Fernández, “Google Colab, una nueva herramienta para IA y Data Analysis,” https://www.tokioschool.com/noticias/google-colab/, Agosto 2022, (Accessed on 06/18/2024).

[26] A. Kolmogorov, “Logical basis for information theory and probability theory,” *IEEE Transactions on Information Theory*, vol. 14, num. 5, pp. 662–664, 1968.

[27] W. Ryan y S. Lin, *Channel Codes: Classical and Modern*. Cambridge university press, 2009.

[28] A. B. Fontaine y W. W. Peterson, “On Coding for the Binary Symmetric Channel,” *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 77, num. 5, pp. 638–647, 1958.

[29] T. M. Cover, *Elements of Information Theory*. John Wiley & Sons, 1999.

[30] R. Gallager, *Information Theory and Reliable Communication*, ser. Courses and lectures. Wiley, 1968. [En línea]. Disponible: https://books.google.com.ec/books?id=Uc3uAAAAMAAJ

[31] S. Lin y D. Costello, *Error Control Coding: Fundamentals and Applications*, ser. Computer applications in electrical engineering series. Prentice-Hall, 1983. [En línea]. Disponible: https://books.google.com.ec/books?id=autQAAAAMAAJ

[32] R. J. McEliece, *The Theory of Information and Coding: Student Edition*, ser. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2004.

[33] R. W. Hamming, “The Art of Doing Science and Engineering: Learning to Learn,” 1998.

[34] G. Sanderson. (2020, September) Pero ¿qué son los códigos hamming? El origen de la corrección de errores. 3Blue1Brown. [En línea]. Disponible: https://www.youtube.com/watch?v=X8jsijhllIA&t=683s

[35] A. Viterbi, “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm,” *IEEE Transactions on Information Theory*, vol. 13, num. 2, pp. 260–269, 1967.

[36] L. Lumburovska, A. Popovska-Mitrovikj, y V. Bakeva, “Comparison of Convolutional Codes and Random Codes Based on Quasigroups for transmission in BSC,” *Security & Future*, vol. 5, num. 3, pp. 98–101, 2021.

[37] D. Gligoroski, S. Markovski, y L. Kocarev, “Error-Correcting Codes Based on Quasigroups,” in *2007 16th International Conference on Computer Communications and Networks*, 2007, pp. 165–172.

[38] A. Popovska-Mitrovikj, S. Markovski, y V. Bakeva, “Increasing the Decoding Speed of Random Codes Based on Quasigroups,” *ICT innovations*, pp. 93–102, 2012.

[39] J. J. Bryson, “La última década y el futuro del impacto de la IA en la sociedad | Openmind,” https://www.bbvaopenmind.com/articulos/la-ultima-decada-y-el-futuro-del-impacto-de-la-ia-en-la-sociedad/, Febrero 2019, (Accessed on 06/23/2024).

[40] R. R. Abril, “Retropropagación del gradiente • Un artículo de LMO,” https://lamaquinaoraculo.com/deep-learning/la-retropropagacion-del-gradiente/, (Accessed on 06/05/2024).

[41] M. Minsky y S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 09 2017. [En línea]. Disponible: https://doi.org/10.7551/mitpress/11301.001.0001

[42] E. C. Fernández, “Tipos de deep learning: Todo lo que debes saber | tokio school,” https://www.tokioschool.com/noticias/tipos-deep-learning/, Marzo 2023, (Accessed on 06/23/2024).

[43] C. Arana, “Redes neuronales recurrentes: Análisis de los modelos especializados en datos secuenciales,” https://www.econstor.eu/bitstream/10419/238422/1/797.pdf, Junio 2021, (Accessed on 06/05/2024).

[44] M. R. B. de Quirós, “Análisis de líneas de costa con redes neuronales LSTM,” https://openaccess.uoc.edu/bitstream/10609/119692/6/mrivasbeTFM0620memoria.pdf, Junio 2020, (Accessed on 06/17/2024).

[45] R. Dey y F. M. Salem, “Gate-variants of Gated Recurrent Unit (GRU) neural networks,” in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 1597–1600.

[46] “Redes neuronales profundas - Tipos y Características - Código Fuente,” https://www.codigofuente.org/redes-neuronales-profundas-tipos-caracteristicas/, Abril 2019, (Accessed on 06/18/2024).

[47] T. O’Shea y J. Hoydis, “An Introduction to Deep Learning for the Physical Layer,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, num. 4, pp. 563–575, 2017.

[48] M. Ibnkahla, “Applications of neural networks to digital communications – a survey,” *Signal Processing*, vol. 80, num. 7, pp. 1185–1215, 2000. [En línea]. Disponible: https://www.sciencedirect.com/science/article/pii/S016516840000030X

[49] T. J. O’Shea, K. Karra, y T. C. Clancy, “Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention,” in *2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2016, pp. 223–228.

[50] E. Nachmani, Y. Beery, y D. Burshtein, “Learning to Decode Linear Codes Using Deep Learning,” 2016.

[51] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, y Y. Be'ery, "Deep Learning Methods for Improved Decoding of Linear Codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, num. 1, pp. 119–131, 2018.

[52] S. Liao, C. Deng, M. Yin, y B. Yuan, "Doubly Residual Neural Decoder: Towards Low-Complexity High-Performance Channel Decoding," 2021.

[53] F.-L. Luo, *Channel Coding with Deep Learning*, 2020, pp. 265–285.

[54] R. Vargas, A. Mosavi, y L. Ruiz, "Deep Learning: A Review," 2017.

[55] T. Elliott, "The State of the Octoverse: Machine Learning - The Github Blog," https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/, Enero 2019, (Accessed on 04/03/2024).

[56] C. D. Costa, "Best Python Libraries for Machine Learning and Deep Learning | by Claire D. Costa | Towards Data Science," https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c, Marzo 2020, (Accessed on 04/03/2024).

[57] D. R. Center, "Qué es Pytorch: una guía completa," https://developer.oracle.com/es/learn/technical-articles/what-is-pytorch, Mayo 2022, (Accessed on 06/23/2024).

[58] P. A. Parellada, "Deep Learning Techinques for LDPC shortening," https://upcommons.upc.edu/handle/2117/334546, Agosto 2020, (Accessed on 06/04/2024).

[59] Qinbobai, "Algoritmo de estimación de canales basado en aprendizaje profundo a lo largo del tiempo Canales de desvanecimiento selectivo | revistas y revistas ieee | exploración ieee," https://ieeexplore.ieee.org/abstract/document/8847452, Marzo 2020, (Accessed on 06/04/2024).

[60] V. Taranalli, "Github - veeresht/Commpy: Digital Communication with Python," https://github.com/veeresht/CommPy, Octubre 2022, (Accessed on 06/11/2024).

[61] F. Tarrés y M. Cabrera, "Codificación de canal II: códigos convolucionales," 2012.

[62] P. David, “Github - daulpavid/pyturbo: A simple reference implementation of turbo codes for error correction in Python.” https://github.com/DaulPavid/pyturbo, October 2018, (Accessed on 06/10/2024).

[63] D. P. Kingma y J. Ba, “Adam: A Method for Stochastic Optimization,” 2017.

[64] “automeris.io: AI assisted data extraction from charts using WebPlotDigitizer,” ht tps://automeris.io/, (Accessed on 06/18/2024).

[65] Alvarez-Bacuilima, “Github - tesisdeepcodeucuenca/tesis-deepcode-ucuenca: Código utilizado para la tesis “Uso de Deep Learning para la codificación y decodificación en canales de una sola vía.”,” https://github.com/TesisDeepcodeUcuenca/Tesis-Deepcode-Ucuenca, Junio 2024, (Accessed on 06/23/2024).

[66] FAIR, “Nevergrad - A gradient-free optimization platform — nevergrad documentation,” https://facebookresearch.github.io/nevergrad/, (Accessed on 06/17/2024).

Anexos

Anexo A: Repositorio de implementaciones

En [65], se presenta el enlace a un repositorio de Git Hub donde se muestra cada uno de los códigos desarrollados en el transcurso de este trabajo de titulación.

La estructura del repositorio es la que se presenta en el siguiente mapa:

```
Repositorio/
├── README.md
├── Códigos convencionales/
│   ├── Codigos_Hamming_(15,_11)_y_datos_crudos.ipynb
│   ├── Convolucional_1_2.ipynb
│   ├── Convolucional_1_3.ipynb
│   ├── Códigos_de_repetición.ipynb
│   └── Turbo_códigos.ipynb
├── Experimentos/
│   ├── Con feedback/
│   │   ├── Experimento_4.ipynb
│   │   └── Experimento_5.ipynb
│   └── Sin feedback/
│       ├── Experimento_1.ipynb
│       ├── Experimento_2.ipynb
│       └── Experimento_3.ipynb
└── Pruebas/
    ├── Graficar_BER_VS_PROBABILIDADES.ipynb
    ├── Graficar_BER_por_Bit.ipynb
    └── Probar_Modelo_enviar_1_y_0.ipynb
```

En función de este mapa se puede encontrar cada experimento debidamente comentado, incluyendo la implementación de códigos convencionales y pruebas realizadas para cada programa desarrollado.