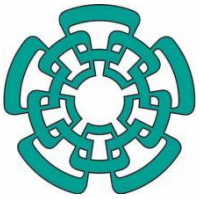


CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN DE COMUNICACIONES

“Decodificación de códigos polares mediante técnicas de aprendizaje automático”

TESIS

Que presenta

JESÚS ÁNGEL SÁNCHEZ RODRÍGUEZ

Para obtener el grado de

MAESTRO EN CIENCIAS

EN LA ESPECIALIDAD DE INGENIERÍA ELECTRICA

Directores de tesis:

Dr. Manuel Mauricio Lara Barrón

Dra. Ana María Antonia Martínez Enríquez

Ciudad de México Abril, 2022

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (Conacyt) por otorgarme apoyo económico durante mi estudio de Maestría por medio del programa de Becas Nacionales.

Agradezco a la Secretaría de Ciencia, Tecnología e Innovación de la Ciudad de México por el apoyo económico brindado a través del proyecto SECTEI/206/2022.

Agradezco al Centro de Investigación y de Estudios Avanzados del IPN (Cinvestav-IPN) por la educación y conocimientos de calidad brindados a lo largo de mi estudio de posgrado.

Doy gracias a la Sección de Comunicaciones del Departamento de Ingeniería Eléctrica y al Departamento de Computación del Cinvestav-IPN unidad Zacatenco por darme la oportunidad de ser parte de esa gran comunidad de profesionales apasionados por el desarrollo científico del país.

Agradezco infinitamente a mis asesores de tesis Dr. Mauricio Lara y Dra. Ana Maria Martínez por el gran esfuerzo que hicieron para ayudarme a concluir este trabajo de tesis, por su amabilidad, consejos y cursos de excelencia.

Le doy gracias a mis profesores Dr. Aldo Orozco, Dra. Giselle Galván, Dr. Amílcar Meneses, Dr. José Rodríguez, Dr. Mario Siller y Dra. Susana Ortega por las invaluables clases que impartieron, por la satisfacción que sentí por aprender algo nuevo y las lágrimas que me costaron adquirir ese conocimiento.

También quiero agradecer a mi madre Magdalena, a mi abuelo Miguel y a mi hermano Aarón por el amor que me dan, por apoyarme en todo momento, siempre confiar en mí y aguantarme en mis momentos de mayor estrés. A mi novia Sarai Fernández, no tengo palabras para describir el profundo amor que siento por ti. A mis tíos Abundio, Andrés y Alejandro, que, poniendo el ejemplo, me enseñaron valores que me llevaron hasta este punto. A Armando Soberanis que con cariño siempre ha estado al pendiente de mí y mi familia. A mi amigo Antonio Ramírez y su familia por todas las risas y por brindarme ese abrazo caluroso que siempre se requiere.

Finalmente, quiero agradecer a toda la comunidad científica por hacer de este mundo un lugar mejor.

Resumen

Para establecer una comunicación entre dispositivos, se requiere que el mensaje que atraviesa un medio no sea modificado por este a tal grado que el receptor lo confunda o no lo entienda. La técnica que se emplea para proteger al mensaje se llama codificación de canal y consiste en agregar bits de manera estructurada al mensaje digital original de tal suerte que el receptor, mediante operaciones matemáticas, es capaz de estimarlo a pesar de que el canal ruidoso lo haya dañado.

En la tecnología en desarrollo 5G, la necesidad de protección, rendimiento y baja latencia se hace aún mayor que con las cuatro generaciones de comunicación móvil anteriores. Por tal motivo, para la información de control, se opta por utilizar códigos polares, debido a que estos códigos logran asintóticamente la capacidad simétrica del canal discreto sin memoria con entrada binaria. Los métodos probabilísticos de decodificación, a pesar de ser precisos, son procesos en serie y no se cumple con el requerimiento en cuanto al tiempo de decodificación.

En este trabajo de tesis se estudia el desempeño de una red neuronal completamente conectada para realizar la tarea de la decodificación. Esta estructura es elegida debido a que el tiempo de cómputo es considerablemente menor a otro tipo de estructuras de red. Además, existe cierto grado de paralelización en el proceso. Con la configuración encontrada, usando 20000 palabras de código con tamaño de 32 bits, tasa de codificación igual a 1/2 y modulación BPSK que atraviesan un canal AWGN a un SNR de 5 dB, se logra una proporción de error de trama (FER) de alrededor de 0.93 con un tiempo (dependiente de la plataforma de simulación) cercano a 0.0004 s por palabra de código. El método probabilístico que se usó para comparación fue el *Fast Simplified Successive Cancellation* que logra una FER de 0.99 con un tiempo de 0.0006 s por palabra de código. Por lo tanto, la estructura de red neuronal completamente conectada, aunque más veloz, no logra una precisión aceptable para ser implementada en un sistema 5G real.

Abstract

In order to establish communication between devices, it is necessary that the message passing through a medium is not modified in such a way that the receiver confuses it or does not understand it. The technique used to protect the message is called channel coding and consists in adding bits in a structured manner to the original digital message in such a way that the receiver, by means of mathematical operations, is able to estimate it even though the noisy channel has damaged it.

In the developing 5G technology, the need for protection, throughput and low latency becomes even greater than in the previous four generations of communications. For this reason, polar codes are chosen because these codes asymptotically achieve the symmetric capacity of the discrete memoryless channel with binary input. Probabilistic decoding methods, although accurate, are serial processes and do not achieve the decoding time requirement.

In this thesis work, the performance of a fully connected neural network is studied to perform the decoding task. This structure is chosen because the computation time is considerably less than other types of network structures. In addition, there exists some degree of parallelization in the process. With the configuration found, using 20000 code words with 32-bit size, coding rate equal to 1/2 and BPSK modulation traversing an AWGN channel at an SNR of 5 dB, a frame error rate (FER) of about 0.93 is achieved with a time (depending on the simulation platform) close to 0.0004 s per code word. The probabilistic method used for comparison was Fast Simplified Successive Cancellation which achieves a FER of 0.99 with a time of 0.0006 s per code word. Therefore, the fully connected neural network structure, although faster, does not achieve acceptable accuracy to be implemented in a real 5G system.

CONTENIDO

LISTA DE FIGURAS.

LISTA DE TABLAS.

GLOSARIO.

- *Backpropagation*: Algoritmo principal de propagación del error de ajuste en las neuronas que conforman una red.
- *Batch*: Sub-conjunto de datos de entrada a la red neuronal.
- *Belief propagation*: Algoritmo de decodificación de propagación de creencias.
- *Bit-reversal*. Inversión de bits.
- *Epoch*: Recorrido del conjunto de datos en su totalidad a través una red neuronal.
- *Fast Simplified Successive Cancellation*: Decodificador rápido simplificado de cancelación sucesiva.
- *Framework*:
- *Non-AWGN*. Canal distinto al AWGN.
- *Overfitting*: Fenómeno que se presenta en las redes neuronales. Ocurre cuando el conjunto de datos de entrenamiento no representa al universo de datos a estimar o el entrenamiento ha sido demasiado extenso y la red neuronal "memoriza" dicho conjunto y no es capaz de estimar datos nuevos.
- *Time step*: Paso de tiempo.
- *Underfitting*: Fenómeno que se presenta en las redes neuronales. Ocurre cuando el conjunto de datos de entrenamiento no es lo suficientemente grande o la topología de la red neuronal es demasiado pequeña para lograr una estimación del universo de datos.

ACRÓNIMOS:

- ADAM: Estimación adaptativa de momento. Por sus siglas en inglés: *Adaptive moment estimation*
- API: Interfaz de programación de aplicaciones. Por sus siglas en inglés: *Application programming interface*.

- AR: Aprendizaje reforzado.
- AWGN: Ruido blanco aditivo gaussiano. Por sus siglas en inglés: *Additive white gaussian noise*.
- BCE: Entropía binaria cruzada. Por sus siglas en inglés: *Binary cross entropy*.
- BCH: Códigos Bose–Chaudhuri–Hocquenghem
- B-DMC: Canal binario discreto sin memoria. Por sus siglas en inglés: *Binary discrete memoryless channel*.
- BER: Proporción de error de bit. Por sus siglas en inglés: *Bit error rate*.
- BP: Algoritmo de decodificación de propagación de creencias.
- BPSK: Modulación por desplazamiento de fase. Por sus siglas en inglés: *Binary phase shift keying*.
- C: Capacidad de canal.
- CRC: Verificación cíclica de redundancia. Por sus siglas en inglés: *Cyclic redundancy check*.
- eMBB: Banda ancha móvil mejorada. Por sus siglas en inglés: *enhanced Mobile Broadband*.
- FER: Proporción de error de trama. Por sus siglas en inglés: *Frame error rate*.
- k: Tamaño del mensaje en bits.
- LDPC: Verificación de paridad de baja densidad. Por sus siglas en inglés: *Low Density Parity Check*.
- LTE: Evolución a largo plazo. Por sus siglas en inglés: *Long term evolution*.
- MALM: Modelo de meta-aprendizaje agnóstico. Por sus siglas en inglés: *Model-Agnostic Meta Learning*.
- MAP: Máximo a posteriori.

- mMTC: Comunicación masiva entre máquinas. Por sus siglas en inglés: *massive Machine Type Communication*s.
- ML: Máxima verosimilitud. Por sus siglas en inglés: *Maximum likelihood.*
- N: Tamaño de la palabra de código en bits.
- NADAM: Estimación adaptativa de momento de Nesterov. Por sus siglas en inglés: *Nesterov adaptive moment estimation*.
- p: Porcentaje.
- PMC: Perceptrón multi-capa.
- PNN: Red neuronal dividida. Por sus siglas en inglés: *Partitioned neural network*.
- R: Tasa de transmisión de información.
- Rc: Tasa de codificación.
- RMSprop: Propagación de la raíz cuadrada media. Por sus siglas en inglés: *Root mean square propagation*.
- RN: Red neuronal.
- RNC: Red neuronal convolucional.
- RNR: Red neuronal recurrente.
- SC: Cancelación simple.
- SCL: Cancelación simple por lista.
- SNR: Relación señal a ruido.
- TBCC: *Tail-biting convolutional codes.*
- URLLC: Comunicación ultra confiable de baja latencia. Por sus siglas en inglés: *Ultra Reliable Low Latency Communications.*

INTRODUCCIÓN.

La última propuesta del estándar de comunicación móvil 5G, lanzada en 2019 [3GPP, 2019], considera un cambio con respecto al estándar 4G-LTE en cuanto a la codificación de canal empleada, pasando de turbo códigos a códigos Polares y LDPC (*Low Density Parity Check*). Estos dos últimos son tomados en cuenta de manera seria desde el "3GPP RAN1 meeting #87 final report" en 2016 [3GPP, 2016] debido a las exigencias de confiabilidad y baja latencia con las que cumple. En el estándar 5G se especifica el uso de códigos polares para la información de control y los datos del usuario son codificados con códigos LDPC [3GPP, 2019].

Buscando mejorar las condiciones con las que se establece la comunicación, se ha optado por investigar maneras de realizar la codificación y decodificación de canal distintas al método clásico. Los logros que se persiguen son, por ejemplo, alto desempeño en canales *Non-AWGN* y reducir la complejidad computacional en la decodificación, que resulta en el uso de menos energía en dispositivos móviles. La mayoría de los esquemas de decodificación adolecen de la primera característica en especial.

Investigadores desde la década pasada han realizado avances en el uso de técnicas de aprendizaje automático, en particular redes neuronales y aprendizaje reforzado, para distintos decodificadores, alcanzando los puntos mencionados anteriormente con desempeños en cuanto a la SNR cercanos al óptimo. Lo anterior resulta en la motivación de este trabajo, el cual será el estudio y simulación de técnicas de aprendizaje automático para la decodificación de códigos polares. La elección de este tipo de códigos está basada en el reciente interés que ha adquirido dentro de la comunidad científica y su aplicación en las técnicas modernas de comunicación 5G y posteriores.

CAPÍTULO 1 REVISIÓN DE TÉCNICAS DE APRENDIZAJE AUTOMÁTICO APLICADAS A LA CODIFICACIÓN DE CANAL.

1.1. JUSTIFICACIÓN.

Dadas las necesidades actuales de comunicación entre personas y dispositivos, se vuelve fundamental la investigación en temas que empujen el desarrollo de los nuevos sistemas de comunicación. Cuando observamos las emocionantes aplicaciones de la tecnología 5G y posteriores, nos damos cuenta de que en todos los casos se requiere cumplir en cierta medida con las siguientes características.

- Gran tasa de transmisión de datos

Alrededor de 20 Gbs en 5G y del orden de los Tbs en 6G para la descarga. La necesidad actual de la tecnología 5G es 1000 veces más grande que la ofrecida por 4G. En un escenario de baja demanda para la red 5G se desea contar con el 5% de la tasa individual de transmisión de datos de 1 Gbs. El requerimiento anterior es 100 veces más grande que para 4G [Andrews, 2014].

- Baja latencia

En 5G se requiere que la latencia en datos de usuario sea de 0.5 ms, para 6G este valor es menor a 0.1 ms. La latencia de datos de control debe ser menor a 10 ms con 5G y menor a 1 ms con 6G. La latencia en la red 4G para datos de usuario es de alrededor 15 ms [Tariq et al, 2019].

- Bajo consumo energético

La enorme cantidad de datos que se transmiten en la red 5G y posteriores provoca que el consumo energético por byte transmitido se deba reducir en la misma tasa que la transmisión de datos crece [Buzzy et al, 2016].

La medida de cumplimiento de tales requerimientos depende del tipo de aplicación. En el estándar 5G se registran 3 amplias categorías [Popovski et al, 2018]:

- Banda ancha móvil mejorada (eMBB).

Para dispositivos que transmiten con cierto patrón estable por un periodo de tiempo largo. Está pensado para lograr tasas de transmisión más altas con una confiabilidad moderada; las aplicaciones pueden ser enfocadas al entretenimiento, por ejemplo, videojuegos en línea y transmisiones en vivo o

usos más serios como notificaciones continuas a dispositivos en una cadena de producción y comunicación entre vehículos.

- Comunicación masiva entre máquinas (mMTC).

Para transmisión de datos baja y aleatoria de una gran cantidad de dispositivos conectados a una estación base. La aplicación es básicamente crear ciudades inteligentes completamente integradas.

- Comunicación ultra confiable de baja latencia (URLLC).

Este esquema es un híbrido de los dos anteriores: se pueden agendar transmisiones de larga duración como en eMBB pero con una confiabilidad de hasta dos órdenes de magnitud de diferencia; también se pueden agendar transmisiones cortas y aleatorias como en mMTC pero la tasa de transmisión de datos es más grande.

El problema de obtener la comunicación con la calidad que el estándar 5G plantea ha sido estudiado por importantes grupos de investigadores, empresas privadas, universidades, etc. En específico la asociación 3GPP, integrada por asociaciones como "China Communications Standards Association", "The Alliance for Telecommunications Industry Solutions, USA" y "The European Telecommunications Standards Institute", se dio a la tarea de desarrollar y proponer el modelo 5G en [3GPP, 2019]. En ese documento se presenta la manera tentativa de cumplir con las necesidades en la comunicación 5G, explicando cada una de las capas del protocolo de comunicación.

Una de las características preponderantes es una comunicación ultra confiable. Es de suma importancia contar con mecanismos de protección de mensajes, ya que el canal por el cual tales mensajes son enviados provoca daños en estos, resultando en una falla catastrófica. La técnica que nos permite proteger al mensaje por medio de bits de redundancia agregados de manera estructurada es la codificación de canal [Sayood, 2012].

Los modelos de codificación son cuantiosos y no existe un esquema único que resuelva los distintos problemas que devienen de la aplicación específica. En general, en el lado del emisor, el mensaje es cubierto con bits de redundancia y, en el lado del receptor, se implementa un esquema de decodificación que puede detectar y corregir fallas. Por desgracia, la precisión de estos decodificadores viene acompañada con una considerable complejidad.

Para las necesidades que se tenían durante el desarrollo hasta su madurez del estándar 4G-LTE, los tipos de códigos elegidos fueron los Turbo códigos para proteger los datos del usuario y los códigos TBCCs (*tail-biting convolutional codes*) para los datos de control. A pesar de que los Turbo códigos fueron considerados en [3GPP, 2017], esta asociación decidió enfocar sus esfuerzos en los códigos LDPC y Polares [Tahir, 2017].

Los códigos LDPC son utilizados para codificar datos, ya que muestran una mejor eficiencia en la cobertura en zonas amplias, una complejidad menor en su decodificación y hay menor probabilidad de errar en tal proceso de decodificar. Lo anterior en comparación con los códigos Turbo [Hui, 2018].

Por otro lado, los códigos Polares se emplean en los datos de control, dado que se ha demostrado que estos tienen un mejor rendimiento para tamaños de bloque pequeños comparado con los códigos LDPC [Tahir, 2017]. Los casos en donde el tamaño sea realmente corto se prefiere *repetition/block coding* en lugar de LDPC. Este tipo de códigos son considerados en los escenarios eMBB y URLLC [3GPP, 2020].

La decodificación de códigos polares que se ha demostrado ser sub-óptima pero con buenos resultados en cuanto a complejidad, creada por [Tal et al, 2012], consiste en una serie de complejos pasos que provoca que la decodificación no sea un proceso paralelizable. El mismo problema se tiene con el método de decodificación de Cancelación Simple propuesto por el creador de los códigos polares [Arikan, 2009]. En el caso del decodificador con el algoritmo *belief propagation*, el desempeño en cuanto a error de decodificación no es aceptable [Tal et al, 2012].

La paralelización del proceso de decodificación es una característica realmente deseable ya que se puede reducir en buena medida el tiempo en la decodificación [3GPP, 2017]. La anterior es una de las razones que motiva la búsqueda de formas alternas de estimar el mensaje original. Como se verá a continuación, distintos investigadores han puesto su atención en algoritmos de inteligencia artificial para esta tarea, específicamente en el campo del aprendizaje automático [Luo, 2020]. En la siguiente sección se mostrarán algunos trabajos que ejemplifican el avance en la aplicación de técnicas de aprendizaje automático para la decodificación, algunas ya pensadas para

códigos polares. Se explicará a grandes rasgos el modelo que proponen y los resultados reportados en sus artículos.

1.2. ESTADO DEL ARTE.

Se ha demostrado que las redes neuronales pueden aprender métodos de decodificación de códigos estructurados (códigos polares por ejemplo) con confiabilidad cercana a la obtenida con el estimador de máximo a-posteriori (MAP) [Xu et al, 2017], [Bennatan et al, 2018], [Jiang et al, 2019b]. Si se pudiera entrenar con todas las palabras de código existentes, se alcanzaría tal meta. Un gran problema es que, dada la complejidad de entrenamiento y ejecución, que crecen de manera exponencial con el tamaño de la palabra de código (*N*), ha sido poco viable entrenar la red neuronal (RN) con tamaños de bloque necesarios para la tecnología 5G.

Bajo esta realidad, en [Gruber et al, 2017] se propone un decodificador de códigos polares con tamaño de bloque menores a 64 bits. La estructura es un perceptrón multicapa (PMC) completamente conectado con capa de entrada de tamaño *N*, tres capas ocultas de tamaño 128-64-32 con función de activación ReLU, una capa que incluye el ruido AWGN para que la palabra de código modulada sea la entrada a la RN y, por último, una capa de salida tamaño *N* con función de activación Softmax que nos da una predicción del símbolo enviado. Se demuestra que conforme crece el tamaño de las capas ocultas se requiere de menos *epochs* (recorrido del conjunto de entrenamiento completo a través de la RN) para entrenarla debido a que nunca se repite la misma entrada; distinto a otras aplicaciones que sufren de *overfitting* (la RN también aprende las características del ruido). Para la arquitectura elegida se demuestra que el número de *epochs* correcto es 2^{18} con entrenamiento *backpropagation* y optimizador Adam [Kingma et al, 2014] y relación señal a ruido (SNR del inglés *Signal to Noise Ratio*) igual a 1 dB.

Una ventaja con respecto a los decodificadores convencionales es la baja latencia en la decodificación. Una RN completamente conectada realiza la decodificación con un solo recorrido sobre ella y no de manera iterativa como con Cancelación Sucesiva (SC) [Arikan, 2009], Cancelación Sucesiva por lista (SLC) [Tal et al, 2012] y *Belief Propagation* (BP) [Gallager et al, 1962]. Por la estructura de la RN, la decodificación es altamente paralelizable y de baja

complejidad. El mayor inconveniente es la escalabilidad en el tamaño de bloque.

Un método convencional de decodificación polar que presentó mejoras en cuanto a latencia para *N* grande es el SLC dividido [Hashemi et al, 2017]. Consiste en segmentar la gráfica de codificación en sub-gráficas de manera cuidadosa y se decodifica cada parte de forma independiente ya sea por SLC o BP. Siguiendo esta idea, el decodificador de [Gruber et al, 2017] se utiliza para decodificar las particiones mencionadas anteriormente. Una vez entrenado el conjunto de RNs, este es insertado en el método de decodificación BP [Cammerer et al, 2017], de esta manera, la unión de las sub-palabras de código recibidas se hace con las reglas del algoritmo BP, pero las predicciones son realizadas por las RNs. Lo anterior se muestra en la Figura 1.

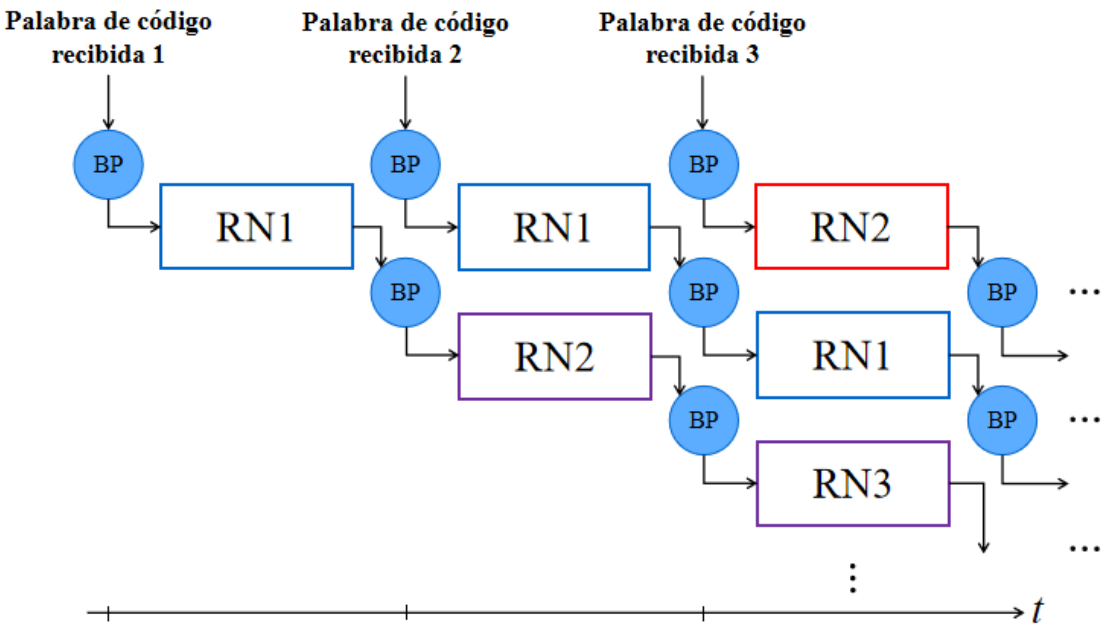


Figura 1. Flujo de la ejecución del decodificador [Cammerer et al, 2017].

Con este decodificador denominado PNN (*Partitioned Neural Network*) se logran resultados similares a SC y BP en cuanto a la proporción de error de bit (BER del inglés *Bit Error Rate*); SLC sigue siendo la mejor opción en este sentido. El tamaño de bloque aún sigue siendo limitado (*N=128*). Está demostrado que la predicción se degrada con divisiones en sub-bloques pequeños, y la degradación depende de la forma de repartir los *k* bits de información entre ellos.

En [Nachmani et al, 2016] también se muestra que es posible optimizar el algoritmo BP con técnicas de aprendizaje profundo, pero ahora aplicado a códigos de corrección Bose–Chaudhuri–Hocquenghem (BCH) con modulación BPSK en un canal AWGN. Un aspecto interesante es que el conjunto de entrenamiento es únicamente la palabra compuesta de ceros en cada bit. Se proponen sustituir 3 funciones que realizan el paso de información sobre el grafo de Tanner en el algoritmo BP. Tales sustituciones forman un conjunto de funciones hiperbólicas pesadas por parámetros que una RN no totalmente conectada con topología definida por el grafo de Tanner debe aprender (con el algoritmo *backpropagation*). De cualquier modo, el método sigue siendo computacionalmente costoso por la naturaleza del algoritmo BP.

En [Lugosch et al, 2017] se sigue un camino parecido pero ahora con el algoritmo BP modificado por la función *min-sum*. Esto resulta en una BER cercana a la presentada por [Nachmani et al, 2016] para códigos BCH con N=64, k=36 *y* k=45 pero mejorando el desempeño para N=127 y k=106. Además, logran reducir la complejidad debido a que el conjunto de parámetros es menor. Por el tipo de código, la modulación y la estructura del decodificador, la red puede ser entrenada con la palabra de código **0**.

Una mejora notable en cuanto a tiempo de ejecución y BER son los reportados por [Xu et al, 2017]. Crean el método de decodificación BP multi-escala con base en la modificación hecha por [Yuan et al, 2014], que consiste en incluir un factor α (calculado de manera experimental) en las funciones de propagación de información de izquierda a derecha en la grafo de factor (generalización del grafo de Tanner) del algoritmo BP. En [Xu et al, 2017] arreglan las ecuaciones que propagan la información del algoritmo BP para colocar un factor β en las funciones de paso de mensaje de derecha a izquierda. Los valores de los factores son obtenidos por medio de una RN con 5 capas ocultas, todas de tamaño 64, capa de salida Sigmoidal y optimizador Adam.

Los investigadores aseguran que el algoritmo es aplicable a cualquier tamaño de bloque, pero solo muestran resultados para códigos polares (64,32) con canal AWGN. Bajo tales condiciones y con únicamente 5 iteraciones lograron una BER similar a la obtenida por 30 iteraciones del algoritmo BP. Esto implica una buena reducción en latencia de decodificación. La RN puede ser incluida en

cualquier variante del algoritmo BP, por ejemplo, en los pensados en los códigos con tamaño de bloque grande.

Con los avances logrados en ese momento, en [Nachmani et al, 2018] pusieron su atención en otra técnica de aprendizaje automático. Desarrollaron un decodificador de códigos lineales que aprende un único parámetro mediante una red neuronal recurrente (RNR) presentando mejores resultados para N grande. El decodificador utiliza la variante del algoritmo BP diseñada por [Dimnik et al, 2009], algoritmo que se realiza con procesos paralelos.

Con una RNR de tamaño 5 entrenada para códigos BCH (63,36) en canales AWGN se llega a 0.6 dB de diferencia con respecto al estimador *maximum likelihood* (ML). Para $N>100$ se logra 1.5 dB de ventaja contra BP; para bloques menores la ganancia es de 1 dB. Lo anterior para BER=10^{-5} [Nachmani et al, 2017].

En [Lyu et al, 2018] realizaron un comparativo entre distintas arquitecturas: perceptrón multicapa, red neuronal convolucional (RNC) y RNR, cuidando los hiper-parámetros como sugiere [Bergstra et al, 2011]. Llegaron a la conclusión de que existe un tamaño N, que denominan de saturación, sobre el cual la red permanece en estado de *underfitting*, es decir, la RN no es capaz de aprender.

Los experimentos fueron realizados con y sin ruido, tomando un porcentaje p de las palabras de código totales para el conjunto de entrenamiento. Para palabras de código de tamaño $N=16$ sin ruido se logra una BER=0 con todas las estructuras mostrando que RNRs y RNCs tienen un mayor poder de aprendizaje. En el caso de $N=32$ la única estructura que logró $BER=0$ fue la RNR, sin importar el porcentaje p. Para palabras de código de tamaño $N=8$ con ruido, todas las estructuras logran la *BER* obtenida con MAP utilizando todas las palabras de código posibles; con p menores el resultado es muy pobre. Con $N=16$ solo la RNR alcanza el desempeño del MAP con $p=100$ %.

Se concluye que N y p determinan qué tan bien se ajustan las RNs y que, para la aplicación de éstas en decodificación de canal, las RNRs son la mejor opción en cuanto a la *BER* a cambio de un entrenamiento y decodificación más lentos; alrededor del doble de tiempo que se necesita para una RNC y cuatro veces más que para un PMC.

Para superar el obstáculo que implica incrementar el tamaño de N, en [Doan et al, 2018] desarrollaron un método similar al decodificador PNN (pensado para

bloques grandes). La diferencia es que se deja de lado el algoritmo BP y se une la información de las múltiples RNs por medio del algoritmo SC, reportando mejores resultados en el sentido de la latencia de decodificación; alrededor de la mitad de *time steps* requeridos por PNN.

En [3GPP, 2011] se propone una verificación cíclica de redundancia (CRC) en conjunto con códigos polares para mitigar el error que surge de tener bloques de tamaño finito. Con el objetivo de obtener beneficios de la estructura paralela del algoritmo BP, en [Doan et al, 2019] diseñaron un algoritmo que unifica las gráficas de factores de BP y CRC. Después, la predicción se realiza ejecutando BP sobre la nueva gráfica. Además, a cada iteración se utiliza el criterio de detención por medio de CRC propuesto por [Ren et al, 2015] para evitar iteraciones innecesarias.

Tal decodificador se modifica con RNs que asignan pesos aprendidos a la gráfica unificada como en [Nachmani et al, 2018]. En [Doan et al, 2019] muestran que se obtiene un menor error de corrección si se comparan ambos esquemas solo por incluir CRC.

Con este nuevo decodificador a una proporción de error de trama (FER) de 10^{-5} se tiene una ventaja de 0.5 dB sobre el decodificador de [Ren et al, 2015] gracias al intercambio que hay de información entre la gráfica de factores de BP y CRC, a diferencia de utilizar CRC únicamente como criterio de paro. Los resultados presentados en [Doan et al, 2019] se obtuvieron para códigos polares (128, 80) y un CRC de 16 bits.

Se ha mostrado en [Kim et al, 2018] que RNs entrenadas pueden decodificar mensajes que atraviesan canales *Non-AWGN*. Para lograrlo se requiere de un conjunto de datos suficientemente grande que contenga información de dicho canal. Si el canal varía, los parámetros aprendidos posiblemente ya no sean los correctos y la red tendrá mayores errores de decodificación.

Con el objetivo de sortear tal inconveniente, en [Jiang et al, 2019] se explica un método de entrenamiento llamado MALM (*Model Agnostic Meta Learning*). Es un modelo con base en el algoritmo del descenso de gradiente que no modifica parámetros únicamente en el entrenamiento, también lo hace en la fase de prueba, es decir, cuando se alimenta a la RN con datos que nunca ha visto y no pertenecen al conjunto de entrenamiento. Lo anterior permite una gran adaptabilidad a variaciones del canal posteriores al entrenamiento.

Como todo método de entrenamiento para una RN, se requiere de una función de pérdida que señala el error de la predicción. En este caso eligieron la función *Binary*

Cross-Entropy (BCE). Después hay una fase denominada de *meta-entrenamiento.* Por medio del descenso del gradiente sobre la función de pérdida BCE, se calculan parámetros de inicialización para el entrenamiento de la RN.

Este modelo fue probado en el decodificador de turbo códigos que usa una RNR propuesto en [Kim et al, 2018]. Se reportó resultados similares en cuanto al BER incluso sin haber entrenado la RN; el desempeño se obtuvo tan solo con la fase de prueba de MALM. Este esquema de entrenamiento puede ser incluido en decodificadores para otros tipos de códigos, aunque estos resultados aun no han sido presentados.

Otra técnica de aprendizaje automático probada para la decodificación de códigos polares es el aprendizaje reforzado (AR). En [Doan et al, 2020], mediante el algoritmo ε*-Greedy* del problema del bandido multi-armado [Sutton et al, 2018], aprenden las permutaciones correctas en la gráfica de factores del algoritmo BP durante la decodificación. Se ha observado que por medio de permutaciones independientes sobre la gráfica se logra reducir el error de corrección del algoritmo BP [Elkelesh et al, 2018].

Después de encontrar las permutaciones, se utiliza el decodificador de [Doan et al, 2019] para aprovechar los beneficios sobre palabras de código polares de tamaño (128,80) con CRC de 16 bits. Gracias a la selección de permutaciones por AR se logran 0.5 dB de ventaja con respecto a [Doan et al, 2019] y 0.125 dB de la selección de permutaciones de manera aleatoria sin latencia adicional con una FER de 10^{-4}.

Se observó en [Tal et al, 2013] que la elección de los N-k bits congelados, que no representan información, impacta en el error de corrección. Un algoritmo que realiza la tarea de identificar dichos bits, por técnicas de aprendizaje reforzado, es el propuesto por [Liao et al, 2020]. Plantean el problema como un juego de recorrido del laberinto con la esperanza del mayor retorno, en este caso un FER pequeño. La penalización del juego se da si el decodificador (SCL o SC)

no identifica la palabra de código. El algoritmo que resuelve el juego es el llamado SARSA(λ) descrito en [Sutton et al, 2018].

Para códigos polares (128,80) casi se iguala la FER que se obtiene con la construcción de [Tal et al, 2013] y decodificador SCL auxiliado de un CRC. La diferencia es que la construcción se realiza eligiendo los bits congelados con un algoritmo secuencial y no ejecutando una gran serie de simulaciones Monte-Carlo, que se convierte en una tarea complicada cuando el tamaño de bloque y la cantidad de bits de información se hacen más grandes.

1.3. PLANTEAMIENTO DEL PROBLEMA.

A pesar de que se cuenta con buenos decodificadores Polares ([Sarkis et al, 2016], [Fan et al, 2016], [Hashemi et al, 2017]), aún se tiene el problema de que los procesos son seriales y la necesidad de baja latencia nos obliga a buscar un método paralelizable para estimar el mensaje original.

Como se observa en el estado del arte, los resultados que publicaron distintos investigadores sugieren que el camino puede estar trazado hacia una solución que tiene como punto central la Inteligencia Artificial. En específico, la solución por medio de redes neuronales se ve conveniente para códigos Polares ya que el tamaño máximo del bloque en el estándar 5G es de 512 bits [3GPP, 2019]; recordar que el tamaño del bloque de entrada a una red neuronal es un problema a considerar en su diseño ya que el entrenamiento puede ser inviable debido a la complejidad del proceso.

Pero ¿los indicadores de desempeño reportados son verdaderos y cumplen con las necesidades del estándar 5G? ¿En realidad la paralelización del proceso de decodificación implica una mejora en cuanto al tiempo de decodificación? Las respuestas a lo anterior, de ser afirmativas, ¿qué aspectos de la red neuronal son la clave para lograr un buen decodificador de códigos Polares?

1.4. OBJETIVOS.

El objetivo de este trabajo es identificar el límite que una RN completamente conectada tiene para estimar un mensaje protegido con códigos polares proveniente de un canal AWGN mediante las siguientes modificaciones a la RN propuesta en [Gruber et al, 2017]:

1. Uso de variantes de la función de activación ReLU.
2. Variación de parámetros en distintos algoritmos de entrenamiento.
3. Topología de la RN.
4. Uso de regularización, normalización y *Dropout* para mejorar el entrenamiento.

CAPÍTULO 2 ASPECTOS TEÓRICOS.

El proceso de comunicación se establece cuando la acción de un dispositivo provoca una reacción de otro. La codificación de canal es una técnica que amplía las posibilidades de tener una comunicación confiable entre dispositivos. Esto se realiza protegiendo la información de modificaciones que ocurren en el trayecto de la fuente al destino.

Considere el sistema de comunicación digital de la Figura 2.

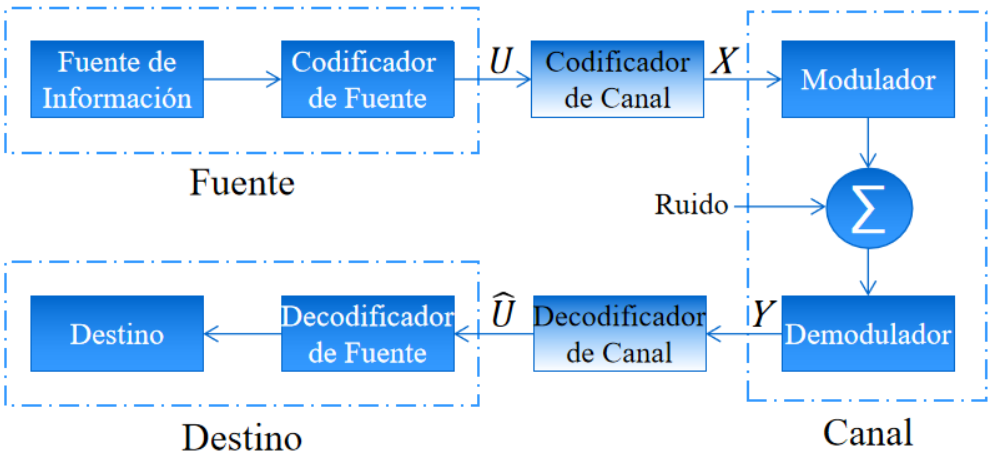


Figura 2. Sistema de comunicación.

La fuente produce $M \in \mathbb{N}$ posibles mensajes U de tamaño $k = \log_2(M)$ bits que se suponen equiprobables e independientes debido a una compresión de datos ideal [Sayood, 2012]. El codificador de canal crea nuevos mensajes X de tamaño N, con $N>k$, agregando N-k bits de redundancia a U de manera estructurada; estos bits serán utilizados por el decodificador de canal para detectar y corregir errores. La tasa de codificación se define como $R_c = k/N$, y representa la fracción de bits de X en el canal que contienen información del mensaje U. Los mensajes X se transmiten al canal, donde pueden ser distorsionados en su camino al destino para producir Y, una versión posiblemente distorsionada de X. El canal consiste en tres partes: el modulador es el medio por el cual creamos la interfaz con el canal físico, el canal físico impone distorsión aleatoria sobre el mensaje y el demodulador es la interfaz con el decodificador. En el demodulador se puede incluir un filtro acoplado y dispositivo de muestreo, dependiendo del tipo de dato requerido. En esta tesis, se considerará un modelo de canal de ruido blanco Gaussiano aditivo AWGN

(del inglés Additive White Gaussian Noise) discreto en tiempo. En este canal, cada muestra del mensaje de salida *Y* equivale al valor de bit correspondiente del mensaje de entrada *X* más un valor aleatorio de ruido.

En el decodificador se hace un estimado $\hat{U}$ del mensaje *U* a partir del mensaje de salida *Y*, el cual contiene parte de la información del mensaje *X* [Lin et al, 1983]. La estimación se hace con base en la naturaleza del canal, la cual queda descrita estadísticamente por medio de la función de densidad de probabilidad condicional *p*(*y*|*x*) llamada probabilidad de transición.

La tasa de transmisión *R* se define como la tasa de transmisión de información o del mensaje, la cual indica la cantidad de bits/s de información de la fuente que son enviados a través del canal, el resto del mensaje es redundancia. Asumiendo un codificador de fuente ideal, todos los bits que surgen de la fuente son información. La capacidad de canal *C*, por otra parte, mide las capacidades de transmisión de información de un canal, y se define como [Sayood, 2012]:

$$C = \max_{p(x)} \left[I(X;Y) \right] \tag{1}$$

donde la información mutua *I(X;Y)* es la diferencia entre la información originalmente en *X* y la información que se pierde debido a las imperfecciones del canal [Sayood, 2012], y el máximo se calcula sobre todas las posibles funciones de masa de probabilidad de entrada *p(x)*.

Shannon demuestra con su segundo teorema [Shannon, 1948] que se puede generar un código libre de errores cuando $R<C$ y el tamaño de bloque es infinito. Así, el objetivo es maximizar la tasa de transmisión del mensaje *R*, sujeta a la limitación del teorema de Shannon.

Un canal quedaría definido por la capacidad que este tiene de transmitir el mensaje, por el tipo de dato de entrada y salida (ya sea discreto o continuo) y la dependencia que esta última tiene con entradas previas [Gallaguer, 1968]. Un buen código es aquel se acerca a la capacidad de determinado canal.

Los esquemas de codificación y decodificación difieren en estructura dependiendo de la aplicación para los que fueron desarrollados. Para comparar estos modelos es necesario tener cierta métrica. Ya que se quiere tener el mínimo error en la decodificación, las métricas usuales son la razón de error de

bit (BER del inglés *Bit Error Rate*) y la razón de error de trama (FER del inglés *Frame Error Rate*); cuyos valores identifican la razón a la que se decodifica de manera equivocada por bit o por bloque respectivamente, y cuya proporción de energía de mensaje y ruido es fija.

2.1. CÓDIGOS POLARES.

Los códigos polares son códigos de bloque que se construyen a partir de una técnica llamada polarización de canal [Arikan, 2009]. Se demostró que por medio de una codificación de este tipo, es posible lograr asintóticamente la capacidad simétrica del canal discreto sin memoria con entrada binaria, B-DMC (del inglés Binary-input Discrete Memoryless Channel). Las características de este canal se encuentran en [Cover et al, 2006].

La polarización de canal consiste en formar un conjunto de $N=2^n$ (con n en los naturales) copias independientes de un canal B-DMC W con probabilidad de transición $p(y|x)$. Lo nuevos canales $W^{(i)}$, con $1<i<N$, tienen la propiedad de que, conforme N crece, un sub-conjunto de estos canales tiene capacidad cercana a 1 bit y el sub-conjunto restante tiene capacidad cercana a 0. Ocurrido este fenómeno, los bits del mensaje de la fuente atraviesan los canales con buena capacidad, mientras que por los canales de baja capacidad se envían los bits de redundancia.

Para que la polarización de canal suceda se requiere de dos etapas: La primera etapa es la combinación, en la que, como su nombre lo indica, se combinan dos copias independientes de un canal B-DMC W: $X \rightarrow Y$, con X y Y alfabetos de entrada y salida, respectivamente. Por ejemplo, la combinación para $N = 2$ se observa en la Figura 3.

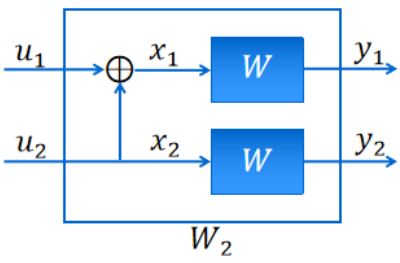


Figura 3. Esquema de codificación polar para N=2.

Formando un nuevo canal W_2: $X^2 \rightarrow Y^2$. Entonces la palabra de código x_1x_2 quedaría formada como se muestra en la Figura 4.

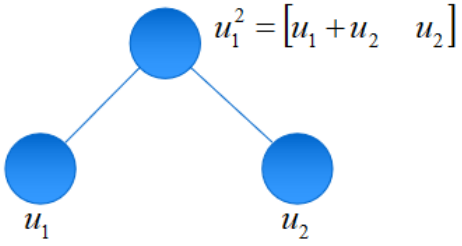


Figura 4. Representación en forma de árbol de codificación polar para $N = 2$.

Es decir, se toma la suma de ambos nodos hijos (los nodos en la parte baja de la Figura 4) para el primer elemento de la palabra de código y el nodo inferior derecho pasa tal cual al segundo elemento de la palabra. Los elementos u_i son bits y la suma es en módulo 2. De la Figura 4 se ve que la probabilidad de transición para el canal W_2 es:

$$p_2(y_1, y_2 \mid u_1, u_2) = p(y_1 \mid u_1 + u_2)\, p(y_2 \mid u_2) \tag{2}$$

Tal relación se cumple ya que las copias de W son independientes entre sí.
Para $N = 4$ se combinan dos canales W_2 de la misma manera para formar el canal W_4. La palabra de código se observa en la Figura 5.

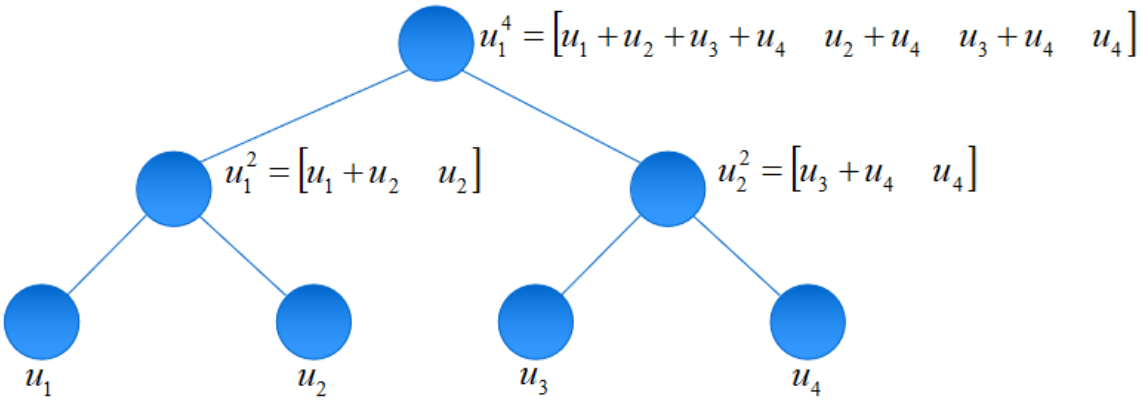


Figura 5. Representación en forma de árbol de codificación polar para $N=4$.

Así, la probabilidad de transición para este canal es:

$$p_4(y_1^2, y_2^2 \mid u_1^2, u_2^2) = p_2(y_1^2 \mid u_1^2 + u_2^2)\, p_2(y_2^2 \mid u_2^2)$$
$$= p(y_1 \mid u_1 + u_2 + u_3 + u_4)\, p(y_2 \mid u_2 + u_4)\, p(y_3 \mid u_3 + u_4)\, p(y_4 \mid u_4) \quad (3)$$

En general, el canal generado en la etapa de combinación, W_N: $X^N \rightarrow Y^N$, tiene probabilidad de transición definida por:

$$p_N(y^N \mid u^N) = p(y^N \mid u^N G_N) = p(y^N \mid u^N B_N F^{\otimes n}) \quad (4)$$

en donde $n = \log_2(N)$. F tiene por nombre matriz kernel de polarización y se define como [Arikan, 2009]:

$$F \equiv \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

El símbolo $\otimes$ indica el producto de Kronecker entre dos matrices y B_N es la matriz *bit-reversal* que provoca una permutación de filas de $F^{\otimes n}$ como se define en [Arikan, 2009].

La siguiente etapa es la división. El canal W_N es separado en N canales de tal manera que cada canal se define como: $W_N^{(i)}$: $X \rightarrow Y^N \times X^{i-1}$ con probabilidad de transición:

$$p_N^{(i)}(y^N, u^{i-1} \mid u_i) \equiv \sum_{u_{i+1}^N \in X^{N-i}} p_N(y^N \mid u^N) \frac{1}{2^{N-1}} \quad (5)$$

en donde el super-índice (i) distingue cada sub-canal. Para y y u, el super-índice determina el tamaño del vector y el sub-índice en u indica la posición del elemento. Tal probabilidad nos permite tener un umbral de decisión en el momento de la decodificación.

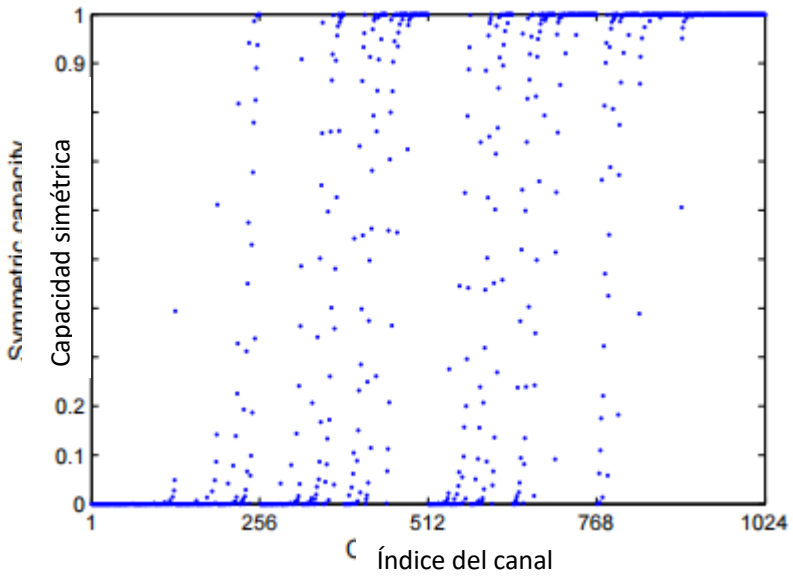


Figura 6. Efecto de la polarización de canal para un canal BEC [Arikan, 2009].

El Teorema 1 de [Arikan, 2009] establece que cuando el tamaño de palabra N tiende a infinito, existe un conjunto de sub-canales que consiguen la capacidad simétrica del canal discreto sin memoria con entrada binaria, B-DMC y otro conjunto con capacidad de canal menor, inclusive, con valor 0. La Figura 6 muestra el efecto de la polarización de canal para un canal binario con borrado (BEC) [Cover y Thomas, 2006]. Se observa que a medida que el índice del canal $W^{(i)}$ aumenta, hay una mayor concentración de canales que logran la capacidad del canal B-DMC. Esta situación es un caso particular del Teorema 1 para un canal BEC. Para otros canales, los índices que alcanzan la capacidad no son necesariamente los mayores.

Cabe mencionar que en la definición de códigos polares, no se especifica los bits de la palabra de código que no llevarán información (bits congelados), es decir, los bits que corresponden a los sub-canales con capacidad de canal igual a 0. A continuación se muestra una lista que ordena los sub-canales de menor a mayor capacidad para 3 valores de N encontrados por simulación.

$N = 8 \rightarrow$ 1, 2, 3, 5, 4, 6, 7, 8.

$N = 16 \rightarrow$ 1, 2, 3, 5, 9, 4, 6, 10, 7, 11, 13, 8, 12, 14, 15, 16.

$N = 32 \rightarrow$ 1, 2, 3, 5, 9, 17, 4, 6, 10, 7, 18, 11, 19, 13, 21, 25, 8, 12, 20, 14, 15, 22,27, 26, 23, 29, 16, 24, 28, 30, 31, 32.

La palabra de código se obtiene con el producto UG_N o con el producto $UF^{\otimes n}$. La diferencia radica en la matriz B_N, que aplica una permutación de filas en la matriz F provocando un cambio de posición de los bits u_i. En adelante se considera que $X = UF^{\otimes n}$.

2.1.1. DECODIFICACIÓN DE CÓDIGOS POLARES.

En la polarización de canal se provoca un efecto parecido a mandar cada bit de la palabra de código por un canal con características distintas, tal efecto se denomina partición de canal lo cual se ilustra en la Figura 7. En realidad, todo el bloque viaja por el mismo canal pero la posición del bit en el bloque determina la cantidad de información que le fue agregada en la etapa de combinación.

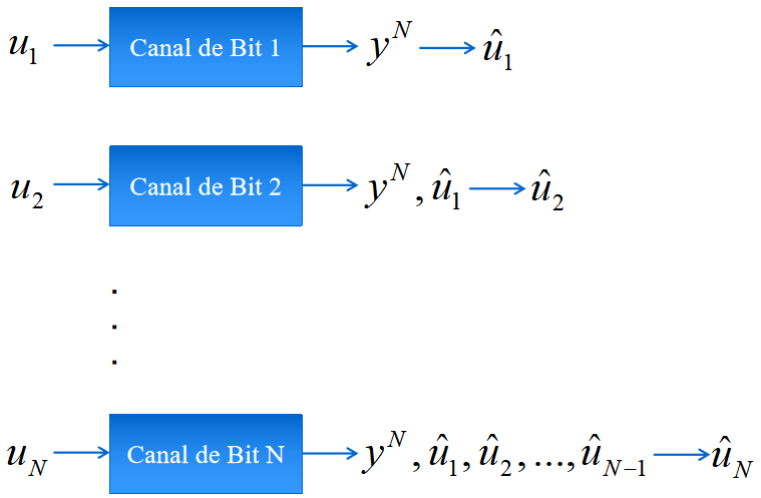


Figura 7. Representación de la partición de canal.

En la decodificación, se tiene el mensaje Y surgido del canal W^N, con esta información se hace una decisión dura para el primer bit de la palabra de código. Enseguida, se toma el mensaje Y y la primera estimación para conseguir el segundo bit. De igual manera, para el tercer bit se toman las predicciones pasadas y se continua de la misma manera hasta obtener todos los elementos de la palabra de código.

Una vez establecidos los bits congelados se tiene una ventaja en la decodificación, ya que en esas posiciones no se realiza decodificación alguna y, por evidentes razones, tal decisión no propaga error.

Suponga que la palabra de código es la que aparece en Figura 4 y que no hay bits congelados. Se recibe el mensaje corrompido $Y = [y_1 \quad y_2]$ y la decisión para el primer bit se toma mediante el método de ML. La elección del bit se puede realizar con otro tipo de funciones que tienen un menor costo computacional; en [Luo, 2020] se propone la ecuación (6), cuyo impacto en la precisión de la estimación es tan pequeño que puede ser despreciable [Arikan, 2009].

$$f(y_1, y_2) = \text{sgn}(y_1)\,\text{sgn}(y_2)\min(|y_1|, |y_2|) \tag{6}$$

.

Si $f < 0$, entonces $\hat{u}_1 = 1$. En caso contrario, $\hat{u}_1 = 0$.

Para el segundo bit: Si $\hat{u}_1 = 0$, entonces se evalúa la función $g(y_1, y_2) = y_1 + y_2$ y se toma la decisión del símbolo de la misma manera, si $g < 0$, entonces $\hat{u}_2 = 1$. De lo contrario, $\hat{u}_2 = 0$ [Giard et al, 2017].

Este es el caso de la repetición, la palabra de código estimada es $\hat{X} = [\hat{u}_2 \quad \hat{u}_2]$ ya que en la codificación se formó la palabra $X = [u_1 + u_2 \quad u_2]$ y u_1 se estimó igual a cero. Entonces el mensaje Y tiene toda la información de u_2, por tal razón se realiza la suma de sus entradas para tomar la decisión.

Si $\hat{u}_1 = 1$, entonces $g(y_1, y_2) = y_1 - y_2$, es decir, retiramos la información de u_1 y se toma la decisión con el mismo criterio. Por último, la palabra de código estimada es $\hat{X} = [\hat{u}_1 + \hat{u}_2 \quad \hat{u}_2]$.

La función g se puede expresar como:

$$g(y_1, y_2, \hat{u}_I) = y_2 + (1 - 2\hat{u}_I)y_1 \tag{7}$$

En donde $\hat{u}_I$ es la estimación del bit del nodo izquierdo.

Para tamaños de bloque mayores, por ejemplo, considere la construcción que aparece en la Figura 5. En este caso el mensaje de salida del canal es $Y = [y_1, y_2, y_3, y_4]$ y nuestro objetivo es encontrar la estimación de la palabra de código $\hat{X} = [\hat{u}_1 + \hat{u}_2 + \hat{u}_3 + \hat{u}_4 \quad \hat{u}_2 + \hat{u}_4 \quad \hat{u}_3 + \hat{u}_4 \quad \hat{u}_4]$.

La decodificación comienza como en el caso de N=2, buscando el valor del bit más a la izquierda.

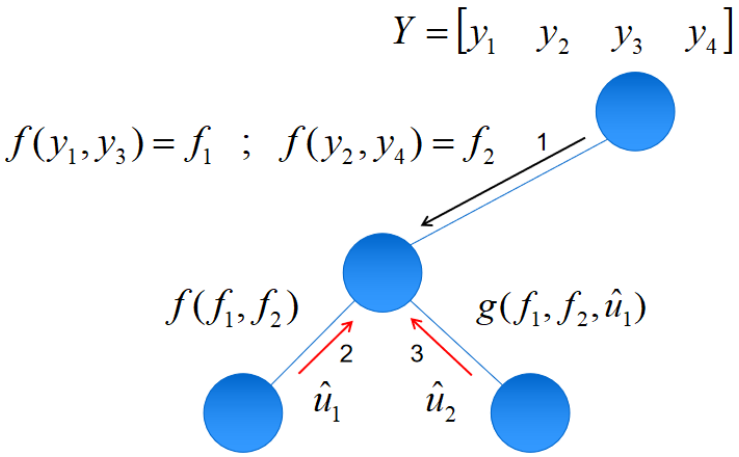


Figura 8. Representación en forma de árbol de la decodificación de códigos polares por cancelación sucesiva.

El paso 1 es propagar la información que se tiene desde el mensaje recibido (Figura 8). Note que el valor f_1 depende de la primera y tercera entrada del mensaje *Y*. Aplicando la función *f* se toma una decisión sobre el símbolo que se acaba de recibir; si tal decisión se realiza de manera correcta, se logra separar la información correspondiente a y_3, esta entrada representa el bit $u_3 + u_4$ dañado por el canal. Lo mismo ocurre para f_2, quitando la información del bit u_4. En el paso 2 se decodifica el primer bit tomando la decisión de nuevo con la función *f*. El paso 3 es decodificar el segundo bit por medio de la función *g* y la estimación anterior y, con eso, el lado izquierdo del árbol de la Figura 5 queda resuelto.

Una vez que se calculan f_1 y f_2, el método para realizar las estimaciones de u_1 y u_2 es idéntico al ilustrado para N=2. Con las estimaciones se consigue más información que será usada para la decodificación del lado derecho del árbol. De igual manera al lado izquierdo, una vez realizada la etapa de propagación de información, se tiene de nuevo la situación del árbol binario, vea Figura 9.

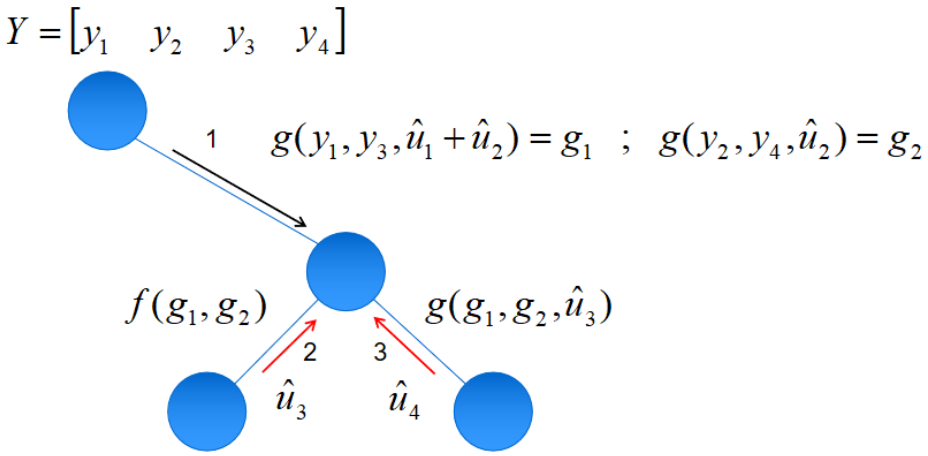


Figura 9. Representación en forma de árbol de la decodificación de códigos polares por cancelación sucesiva.

En g_1 se almacena la información de $u_3 + u_4$ y en g_2 la información de u_4. Con estos valores se realiza la estimación del último par de bits.

De manera general, la decodificación se realiza recorriendo un árbol binario de manera *in order*, es decir, realizar operaciones en el hijo izquierdo, regresar al nodo padre y por último operar sobre el nodo derecho. Si el nodo es hoja, es decir, que no tiene hijos, la operación sería la aplicación de la función *f* o *g* para obtener la estimación. De no ser así, la operación es la propagación de la información en sus distintos casos. Si el nodo que se visita es hijo izquierdo de otro nodo izquierdo, el vector de información es:

$$I_I^j = \left[f\left(f_1^{j-1}, f_{m/2+1}^{j-1}\right) \quad f\left(f_2^{j-1}, f_{m/2+2}^{j-1}\right) \quad \cdots \quad f\left(f_{m/2-1}^{j-1}, f_{m/2}^{j-1}\right) \right] = \left[f_1^j \quad f_2^j \quad \cdots \quad f_{m/2}^j \right] \tag{8}$$

En donde *j* representa la profundidad en el árbol. Para $j = 1$ se tiene $m = N$ y $f_i^0 = y_i$. En etapas posteriores, *m* toma por valor el tamaño del vector I^{j-1}.

Si el nodo que se visita es hijo derecho de un nodo izquierdo, el vector de información es:

$$D_I^j = \left[g\left(f_1^{j-1}, f_{m/2+1}^{j-1}, \hat{x}_1^j\right) \quad g\left(f_2^{j-1}, f_{m/2+2}^{j-1}, \hat{x}_2^j\right) \quad \ldots \quad g\left(f_{m/2-1}^{j-1}, f_m^{j-1}, \hat{x}_{m/2}^j\right) \right]$$
$$= \left[g_1^j \quad g_2^j \quad \ldots \quad g_{m/2}^j \right] \tag{9}$$

Por último, para los nodos que surgen de un nodo derecho, se tienen los vectores:

$$I_D^j = \left[f\left(g_1^{j-1}, g_{m/2+1}^{j-1}\right) \quad f\left(g_2^{j-1}, g_{m/2+2}^{j-1}\right) \quad \ldots \quad f\left(g_{m/2-1}^{j-1}, g_{m/2}^{j-1}\right) \right]$$
$$= \left[f_1^j \quad f_2^j \quad \ldots \quad f_{m/2}^j \right] \tag{10}$$

$$D_D^j = \left[g\left(g_1^{j-1}, f_{m/2+1}^{j-1}, \hat{x}_1^j\right) \quad g\left(g_2^{j-1}, g_{m/2+2}^{j-1}, \hat{x}_2^j\right) \quad \ldots \quad g\left(g_{m/2-1}^{j-1}, g_m^{j-1}, \hat{x}_{m/2}^j\right) \right]$$
$$= \left[g_1^j \quad g_2^j \quad \ldots \quad g_{m/2}^j \right] \tag{11}$$

Que propagan la información al hijo izquierdo y derecho respectivamente.

Para canales B-DMC, la probabilidad de error de bloque para este decodificador es igual a $O(N^{-1/4})$. Si el canal además es simétrico, la inclusión de bits congelados no modifica la probabilidad anterior. La complejidad del algoritmo de decodificación es igual a $O(N \log N)$. Como codificación y decodificación se ejecutan recorriendo un árbol binario, la complejidad para la codificación es la misma [Arikan,2009].

2.2. APRENDIZAJE AUTOMÁTICO.

La meta del aprendizaje automático es imitar o dar información sobre la habilidad de aprendizaje que posee el cerebro humano mediante algoritmos computacionales. Grandes esfuerzos fueron hechos por investigadores como [Turing, 1937], [McCulloch et al, 1943], [Hebb, 1949] y [VonNeumann, 1958] para dar a luz un modelo que asemeje su funcionamiento. Las dos vertientes para llegar a la meta son los modelos monotípicos y genotípicos. En los primeros se describen en su totalidad parámetros, conexión y disposición de los elementos de un sistema completamente enfocado a una tarea en específico. En los segundos, se crean reglas que permiten que los elementos del sistema

ajusten los parámetros del modelo de aprendizaje con base en datos de entrada. En [Rosenblatt, 1963] se sugiere que el segundo paradigma se ajusta mejor a la meta de aprender mediante algoritmos.

Ahora, considere que el cerebro recibe señales desde los "periféricos" del cuerpo humano. Estas señales se interpretan y el cerebro arroja una respuesta que estimula funciones del cuerpo; el ejemplo de una señal es el color observado en un semáforo. La acción que se ejecuta como consecuencia de la entrada anterior es un reflejo de lo aprendido durante algún tipo de entrenamiento. Ya sea que el reglamento de tránsito fue leído o por la experiencia que se adquirió desde la infancia.

Otro ejemplo es el pronóstico del clima de pescadores de alta mar en épocas pasadas. Estas personas han recibido tanta información que la percepción de las señales que otorgan sus "periféricos" les indica si será seguro zarpar. Evidentemente los meteorólogos actuales darán un mejor pronóstico. Esto tiene que ver con la entrada que se le da al cerebro y la calidad del entrenamiento. Para el caso de los meteorólogos, la entrada son datos duros provenientes de sensores y lo ocurrido durante días anteriores y el entrenamiento es una preparación universitaria. De cualquier manera, hay que tener en cuenta que la estimación no siempre será correcta.

Las actividades anteriores (identificar el color del semáforo y la estimación del clima) son tareas de clasificación y regresión que son de gran interés en la rama del aprendizaje automático.

Un método ampliamente utilizado para atacar los problemas anteriores son la redes neuronales, en específico redes de perceptrones multicapa [Géron, 2020]. Las redes de perceptrones multicapa son un esquema que permite aprender parámetros de estimación a partir de un conjunto de datos de entrenamiento que poseen cierta naturaleza, por ejemplo, una distribución de probabilidad. El objetivo es que el sistema pueda aproximar una solución correcta para datos de entrada con la misma naturaleza nunca antes vistos por la RN.

En [Hornik et al, 1989] se demuestra que la redes de perceptrones multicapa son aproximadores universales. Este es el motivo de su relevancia en el campo de la Inteligencia Artificial. Para comprender el funcionamiento de la red se requiere observar la unidad básica llamada neurona. La Figura 10 muestra el esquema de una neurona.

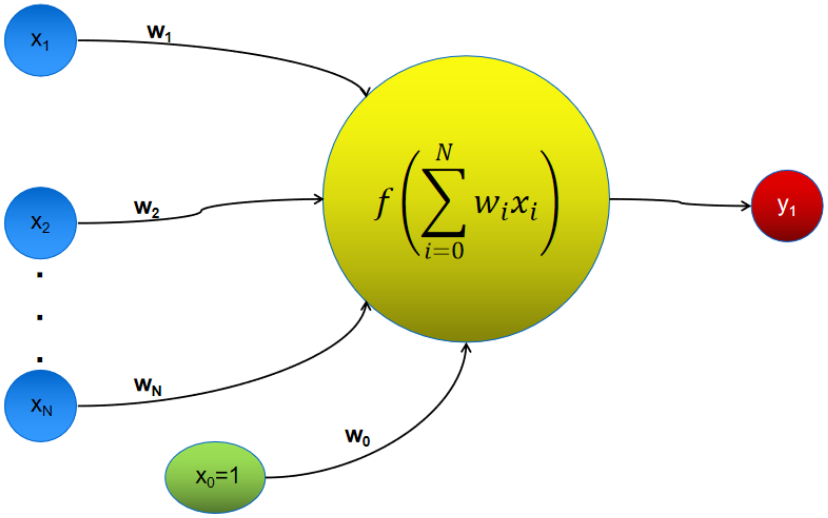


Figura 10. Neurona artificial.

Primero, a los datos de entrada x_i se les asigna un peso w_i, para indicar la relevancia o importancia de cada variable x_i, en algunos casos w_i puede ser cero. Después, se realiza la sumatoria de todos los productos, esto no es más que una regresión lineal de los datos de entrada. Por último, se aplica una función no lineal *f* llamada función de activación, que distorsiona la forma del hiperplano generado por la regresión y con esto se modifica también la salida del perceptrón. Esta distorsión crea zonas de decisión separadas por un umbral. El parámetro w_0 es utilizado para mover el umbral a conveniencia, por ejemplo, colocarlo en el origen. De ahí que su nombre es el parámetro *bias* o de sesgo.

Si los parámetros w_i son elegidos de manera correcta, la neurona puede aproximar de buena manera funciones con zonas de decisión que se separan mediante planos. Los ejemplos clásicos son las estimaciones de las funciones lógicas AND y OR.

Considere el conjunto de datos que se muestra en la Figura 11.

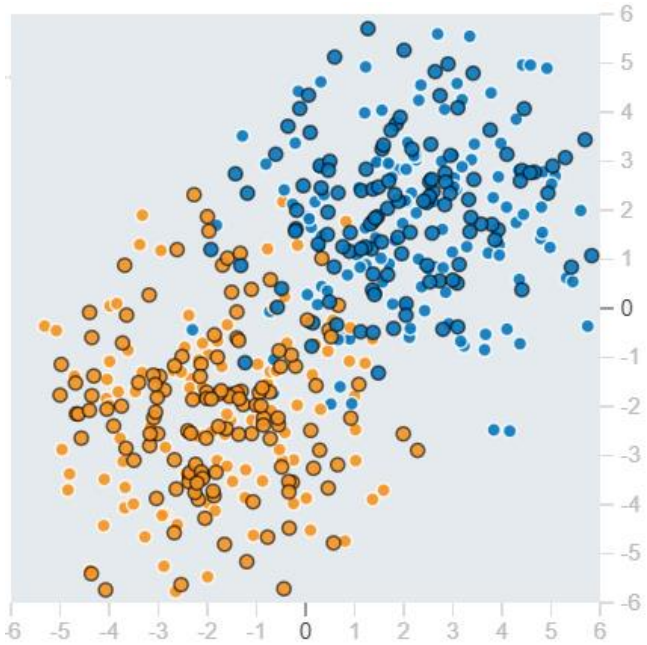


Figura 11. Representación de dos clases en el conjunto de datos. Puntos de distinto color corresponden a una clase diferente.

Una única neurona entrenada es capaz de producir una frontera entre clases, tal que nuevos datos puedan clasificarse en alguna de estas clases. Los datos de ingreso corresponden a las coordenadas de cada punto y la salida se genera en binario.

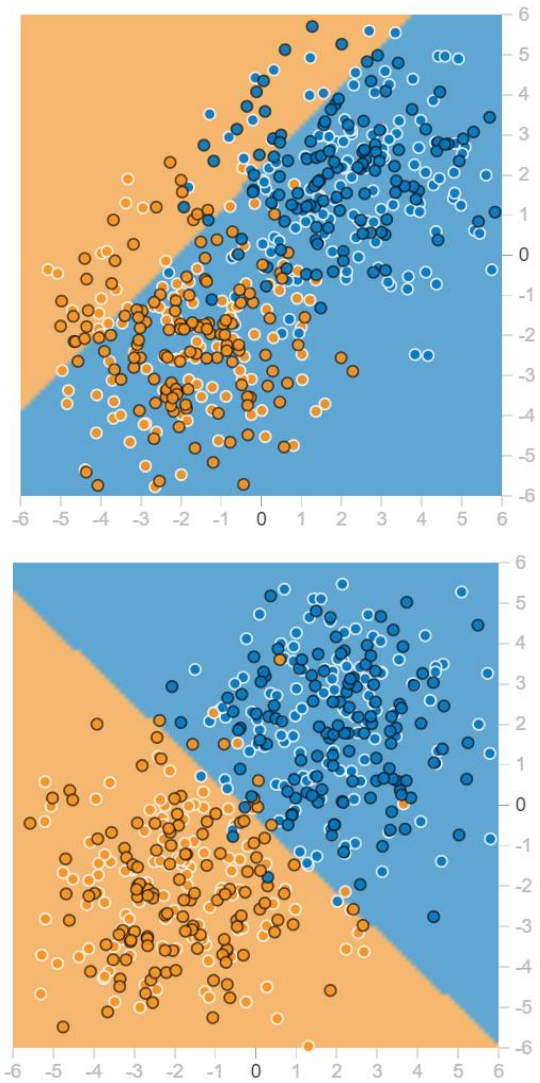
Para marcar la frontera es necesario decidir la función de activación a aplicar. En la Tabla 1 se muestran algunas funciones de activación comunes.

Tabla 1. Funciones de activación comunes [O´Shea et al, 2017].

Nombre	$f(x)$	Rango
Linear	x	$(-\infty, \infty)$
ReLU	$\max(0, x)$	$[0, \infty)$
Tangente Hiperbólica	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$(-1,1)$
Sigmoidal	$\frac{1}{1 + e^{-x}}$	$(0,1)$
Softmax	$\frac{e^x}{\sum_i e^{x_i}}$	$(0,1)$

La efectividad de la función de activación depende de la naturaleza de los datos. Para casos de clasificación binaria la función Linear o Sigmoidal son buenas opciones. En el caso de patrones circulares en los datos, la función tangente hiperbólica puede ser beneficiosa. Si se requiere de una clasificación con múltiples conjuntos, la función Softmax es utilizada ya que da una "probabilidad " de que un dato pertenezca a cierto conjunto [Luo, 2020].

Considere que la función de activación es la Sigmoidal; en la Figura 12.a se puede observar la regiones de decisión que se forman al realizar un *epoch* con parámetros iniciados de manera aleatoria.



a) b)

Figura 12. Fronteras de decisión. a) Epoch=1, b) Error=0.06

Si se reajustan los parámetros de manera correcta se puede obtener una frontera satisfactoria, observe la Figura 12.b. De cualquier manera, es probable que algunos de los datos nunca antes vistos (puntos con color más intenso) no sean clasificados en la clase a la que pertenecen. Dependiendo de la aplicación, la cantidad de veces que los errores ocurren puede ser despreciable.

Como se dijo anteriormente, la función de activación en realidad determina las distintas regiones. Vea la función Sigmoidal de la Figura 13, obtenida mediante la ecuación:

$$f(x_1, x_2) = \frac{1}{1+e^{-x_1-x_2}}$$

Note que los parámetros w_1 y w_2 correspondientes a x_1 y x_2 son iguales a 1 y w_0 es cero. Todos los puntos que evaluados en la función queden sobre la sección de sigmoide apoyada en el plano origen son agrupados en una clase y los que queden en la otra parte del rango pertenecen a la segunda clase.

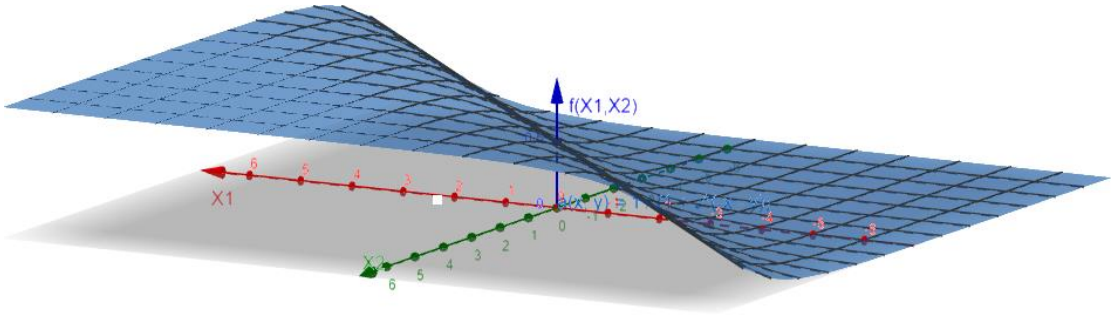
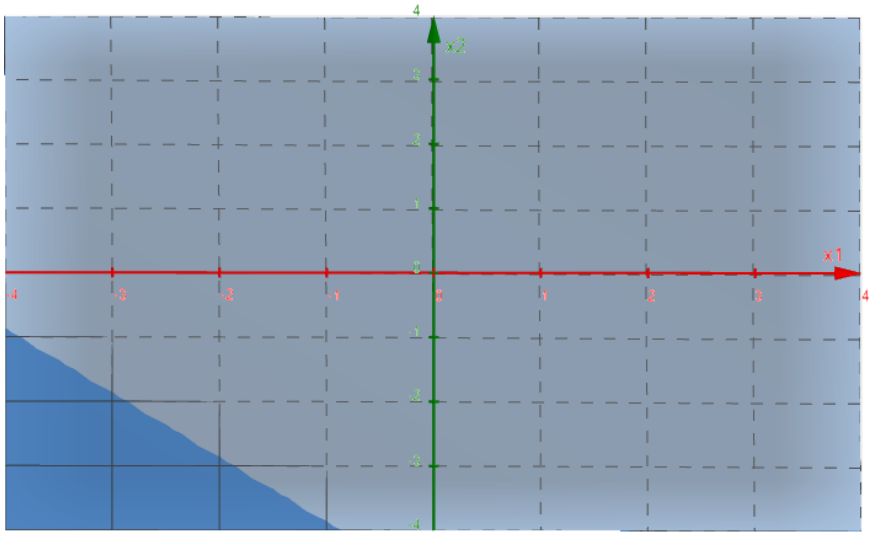


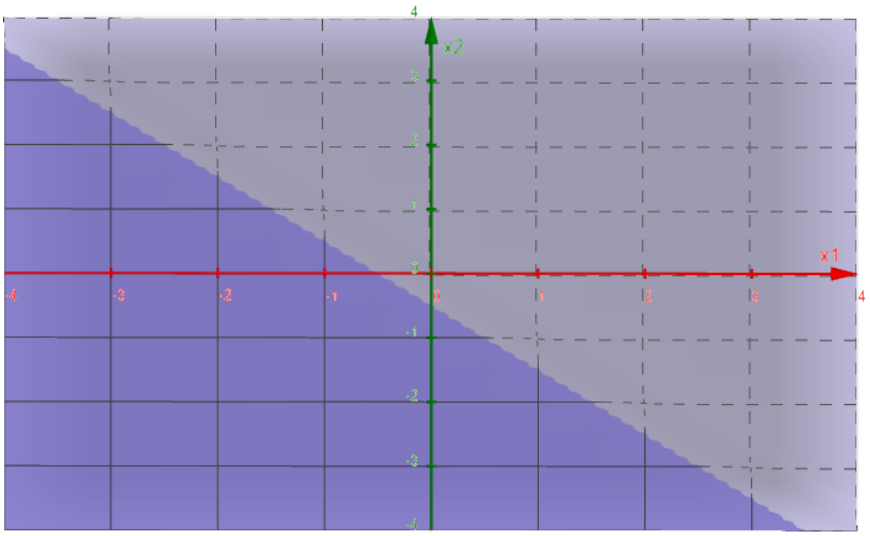
Figura 13. Función Sigmoidal.

La Figura 14 muestra el efecto que tiene el modificar los parámetros de la red.

Una vez elegida la función de activación, la calidad de la estimación dependerá totalmente de la combinación de los parámetros. En Figura 14.a se observa en la esquina superior derecha una región que sobrepasa el plano indicando una mayor cantidad de errores para los datos de Figura 11 en esa región. En Figura 14.b se nota una zona de decisión más ajustada a los datos, aunque no es perfecta, los resultados con estos parámetros pueden ser aceptables.



a)



b)

Figura 14. Modificación de la región de decisión.a) $w_1 = w_2 = 1$. b) $w_1 = w_2 = 10$.

Ahora, observe el conjunto de datos de la Figura 15. Con una neurona no es posible separar los datos en dos categorías, para esta tarea se requiere de estructuras más elaboradas.

Una RN se forma apilando capas de perceptrones. Un perceptrón es la unión de 2 o más neuronas en paralelo, vea la Figura 16. Conforme los datos de entrada se propagan en la red, cada capa consecuente realiza la estimación con base en las regiones de decisión de los perceptrones precedentes. Esto se puede entender como un proceso de limpieza de los datos crudos, de tal manera que los datos de entrada a la capa de salida contengan mayor información para la estimación.

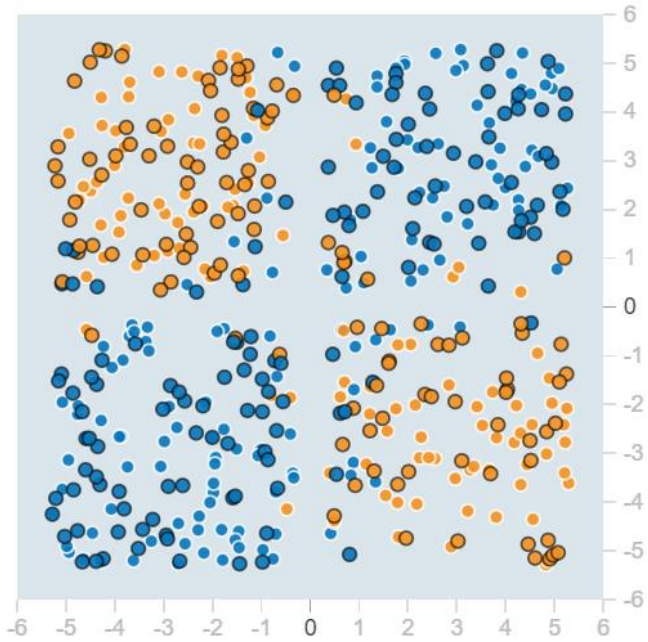


Figura 15. Datos que no se separan con una única división.

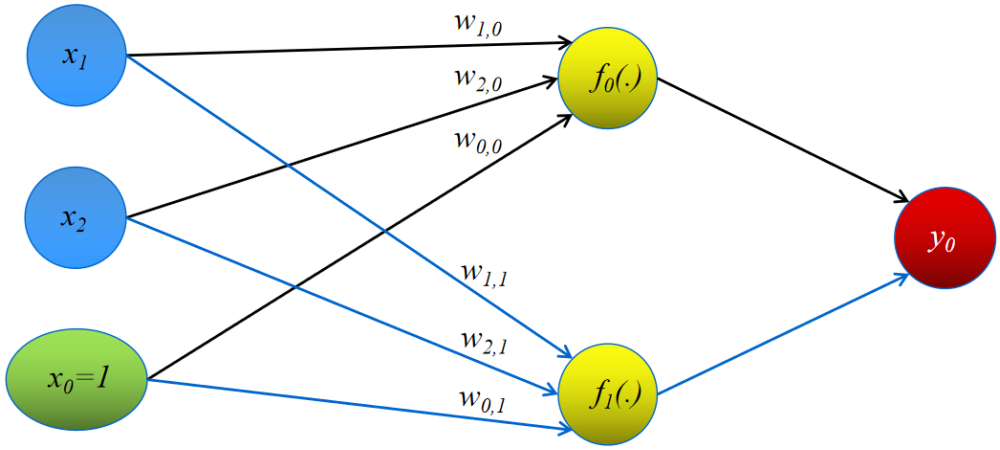


Figura 16. Perceptrón.

La cantidad de perceptrones y la disposición entre ellos es determinante en el funcionamiento de la RN. La frontera trazada en la Figura 17.a proviene de una red con un solo perceptrón conformado por dos neuronas y función de activación Sigmoidal. Con esto ahora se pueden aproximar dos fronteras, una por neurona.

Si se agrega una capa más, idéntica a la anterior, con conexión a ambas neuronas precedentes no se consigue mejora. Esto es debido a que la salida proviene de un perceptrón de tamaño dos y la misma función de activación, por

lo tanto, se mantiene la limitante del tipo de región que las dos neuronas pueden trazar.

La Figura 17.b fue obtenida con una RN de una única capa con el doble de neuronas que el caso anterior; se observa que la desventaja de tener fronteras rígidas se elimina.

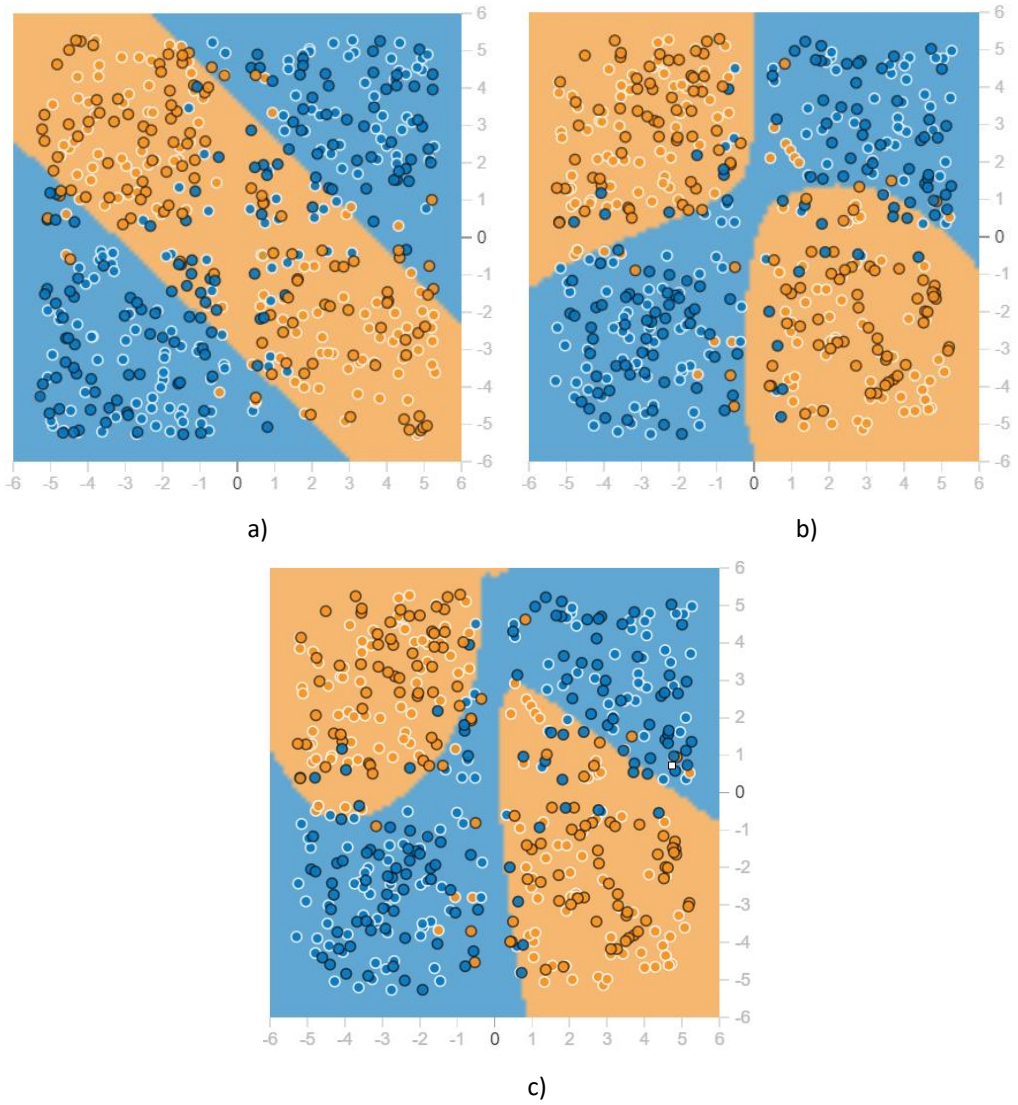


Figura 17. Regiones de decisión.a) Error=0.24.b) Error=0.14. c) Error=0.12.

La siguiente mejora es agregar una capa que influya en la calidad de la estimación. La Figura 17.c se generó con una RN de dos capas completamente conectadas. La segunda capa aprende conocimiento heredado de las 4 neuronas precedentes, que se sabe que arrojan buenos resultados. Con 3 neuronas en la salida es suficiente para eliminar fronteras rectas. Comparando las variantes que aparecen en la Figura 17, la red multicapa parece arrojar el mejor resultado.

En [Minsky, 1959] se muestran las dificultades de ajustar los parámetros de una RN; en ese periodo se hacía prácticamente a fuerza bruta. No fue hasta la publicación de [Rumelhart, 1986] que por fin se conoce un algoritmo eficiente para el entrenamiento llamado *Backpropagation*, método basado en el descenso del gradiente.

Primero, se define una función de coste que representa la diferencia entre la estimación de la RN y el valor original. El valor de la salida depende en cierta proporción de todos los pesos de la red; por medio de la función coste se determina el error que provoca cada uno de los parámetros. La suma ponderada que se realiza en cada neurona también causa errores, por lo tanto, este valor también debe ser considerado en la modificación de los parámetros.

A continuación se muestra en la Tabla 2 dos funciones de costo comunes.

Tabla 2. Funciones de costo [O´Shea et al, 2017].

Nombre	$l(\boldsymbol{u}, \boldsymbol{v})$
MSE	$\|\boldsymbol{u} - \boldsymbol{v}\|_2^2$
Etropía Cruzada Categótica	$-\sum_i u_i \log(v_i)$

La suma ponderada para cada neurona de una capa determinada se puede definir como:

$$z^{(L)} = \sum_{i=1}^{N_L - 1} w_i^{(L-1)} y_i^{(L-1)}$$

En donde L representa el índice de la capa y N_L la cantidad de datos de entrada a la neurona. La salida de la neurona se obtiene aplicando la función de activación a la suma ponderada.

$$y_i^{(L)} = f(z^{(L)})$$

El sub-índice i identifica la salida de cada neurona. Para determinar la cantidad de error, la salida se evalúa en la función de costo.

$$C\left(y_i^{(L)}\right) = C\left(f\left(\mathbf{w}^{(L-1)} \cdot \mathbf{y}^{(L-1)}\right)\right)$$

Dado que el algoritmo de *backpropagation* involucra el descenso del gradiente, es necesario calcular la derivada parcial del coste con respecto a los parámetros para encontrar la combinación de pesos que arroje el mínimo error. Por medio de la regla de la cadena se tiene:

$$\frac{\partial C}{\partial w_i^{(L-1)}} = \frac{\partial C}{\partial y_i^{(L)}}\left(\frac{\partial y_i^{(L)}}{\partial \mathrm{z}^{(L)}}\right)\left(\frac{\partial \mathrm{z}^{(L)}}{\partial w_i^{(L-1)}}\right)$$

El término más a la derecha es el cambio de la sumatoria de la neurona con respecto al peso; este término contiene la cantidad de error en la predicción que se incluye en la neurona. Los otros dos factores restantes se pueden definir como el error que se genera en la evaluación de la función de activación.

$$\delta^L = \frac{\partial C}{\partial y_i^{(L)}} \cdot \frac{\partial y_i^{(L)}}{\partial \mathrm{z}^{(L)}} \tag{12}$$

Si observa el término, la derivada es simplemente el valor de la entrada a la neurona, entonces:

$$\frac{\partial C}{\partial w_i^{(L-1)}} = \delta^L y_i^{L-1} \tag{13}$$

Para relacionar el costo con las capas anteriores a la capa de salida se sigue la misma idea de la regla de la cadena, pero en esta ocasión no aplicamos la derivada de la suma ponderada con respecto al peso, mejor se observa el cambio de esta con respecto a la entrada a la neurona.

$$\frac{\partial C}{\partial \mathbf{w}_i^{(L-2)}} = \left(\frac{\partial C}{\partial y_i^{(L)}} \cdot \frac{\partial y_i^{(L)}}{\partial \mathrm{z}^{(L)}}\right)\left(\frac{\partial \mathrm{z}^{(L)}}{\partial y_i^{(L-1)}} \cdot \frac{\partial y_i^{(L-1)}}{\partial \mathrm{z}^{(L-1)}} \cdot \frac{\partial \mathrm{z}^{(L-1)}}{\partial w_i^{(L-2)}}\right)$$

Reagrupando:

$$\frac{\partial C}{\partial w_i^{(L-2)}} = \delta^L \left(\frac{\partial z^{(L)}}{\partial y_i^{(L-1)}} \cdot \frac{\partial y_i^{(L-1)}}{\partial z^{(L-1)}} \right) y_i^{L-2}$$

Los factores dentro del paréntesis representan el cambio en la suma ponderada de la neurona con respecto a la sumatoria de la capa pasada. La sumatoria de la neurona surge de aplicar la función de activación a la sumatoria anterior y multiplicarla por el peso, entonces se define:

$$\delta^{L-1} = \delta^L w_i^{L-1} \frac{\partial y_i^{(L-1)}}{\partial z^{(L-1)}} \tag{14}$$

Por último, el cambio para este coste con respecto al peso de la capa previa a la salida se define como en la ecuación (13):

$$\frac{\partial C}{\partial w_i^{(L-2)}} = \delta^{L-1} y_i^{L-2}$$

En las capas posteriores se aplica el mismo criterio. Por último, cada uno de los parámetros será modificado de la siguiente manera:

$$w_i^{(L-1)} \leftarrow w_i^{(L-1)} - \lambda \cdot \frac{\partial C}{\partial w_i^{(L-1)}} \tag{15}$$

En donde λ se define como *learning rate* o tasa de aprendizaje. Esta variable permite ajustar la agresividad de modificación de parámetros [Russell et al, 1995].

Como se observa, el algoritmo se ejecuta calculando únicamente 3 ecuaciones, primero la ecuación (12) que involucra dos derivadas, después se obtiene la ecuación (13) con una única multiplicación. Para las capas subsecuentes se calcula la ecuación (14), que requiere de una derivada más, y se aplica la ecuación (13) de nuevo.

Este algoritmo lo podemos aplicar las veces que sea necesario hasta cumplir con cierta métrica. Por ejemplo, los datos se separan en 2 conjuntos, uno para el entrenamiento y uno de prueba. Podemos definir un error aceptable en el conjunto de prueba y en ese momento dejar de aplicar el algoritmo y se conservan los pesos para realizar futuras estimaciones. Además, se puede elegir un tercer conjunto, denominado de validación, conformado por datos nunca antes vistos por la red neuronal y probar la efectividad que esta tiene para trabajar con datos nuevos al concluir el entrenamiento.

Se puede definir el error aceptable en el conjunto de entrenamiento pero se corre el riesgo de sufrir *overfitting*; es decir, los parámetros se ajustan en demasía a los datos de entrenamiento, de manera que ya no le es posible estimar de manera correcta los datos nunca antes vistos por la red. Por otra parte, si el entrenamiento es laxo, se puede sufrir de *underfitting* y la red no será capaz de realizar estimaciones de calidad dado que los parámetros no estarán suficientemente ajustados. Este fenómeno también puede ocurrir gracias a una estructura de red inadecuada para la complejidad en la naturaleza de los datos de entrada [Hastie et al, 2008].

Existen métodos que nos permiten mejorar el funcionamiento de la red. Por ejemplo, se pueden pre-procesar los datos de entrada antes de alimentarlos a la red con el objetivo de hacerlos más representativos. Suponga un conjunto de datos que se dispone de manera circular en el plano. Una entrada más adecuada que las coordenadas podría ser la multiplicación de estas o el cuadrado de las mismas, con esto se aprovechan la geometría del problema.

Otra forma es utilizar una tasa de aprendizaje variable que depende de actualizaciones posteriores o modificar la topología de la RN mediante técnicas de aprendizaje [Alpaydin, 2010]. De esta manera, se obtiene un algoritmo que puede aproximar cualquier tipo de función a un costo computacional asequible para la tecnología moderna.

CAPÍTULO 3 SIMULACIONES

La efectividad de aprendizaje de una RN depende de la estructura y parámetros que la gobierna así como tipo y cantidad de datos de entrada. En esta sección se realizan modificaciones al algoritmo de [Gruber et al, 2017] para tratar de lograr un mejor desempeño para palabras de código pequeñas (N=32 y k=16 como máximo), al igual que identificar el límite que la estructura de una red completamente conectada tiene para resolver el problema de la decodificación de códigos polares.

3.1. PLATAFORMA DE SIMULACIÓN

El lenguaje de programación elegido es Python 3. En [Lutz et al, 2003] se describe como "un lenguaje de programación de propósito general que combina paradigmas procedimentales, funcionales y orientados a objetos". Algunas ventajas de este lenguaje es que está diseñado para que el código sea de fácil lectura con el objetivo de simplificar el mantenimiento del programa, está disponible para diversos sistemas operativos y se cuenta con una gran variedad de bibliotecas con funcionalidades que hacen más sencilla la creación de código, además de que es capaz de invocar bibliotecas propias de otros lenguajes de programación. Con este lenguaje se cuenta con la ayuda de grandes herramientas que facilitan y optimizan el cómputo de algoritmos de aprendizaje automático.

Tensorflow, una biblioteca de código abierto desarrollada por Google para crear y entrenar modelos de inteligencia artificial. Tiene como mayor ventaja la auto-diferenciación, que es necesaria para el cálculo de gradientes en el algoritmo de *backpropagation.* Esta biblioteca permite inicializar los parámetros a entrenar a conveniencia, contiene definiciones de funciones de activación y costo, y por medio de la función *Session* se ejecuta el entrenamiento con la topología de red definida con las funciones mencionadas anteriormente. Otra ventaja es que esta biblioteca cuenta con soporte para el uso de GPUs y ejecución en un entorno distribuido.

Keras, una biblioteca creada en Python que funciona como interfaz de programación de aplicaciones (*API*) para acceder a distintos *frameworks* que desarrollan algoritmos de aprendizaje automático. La ventaja es la facilidad en

la producción de la arquitectura de una red neuronal, ya que se permite crear la red mediante capas predefinidas en la biblioteca. **Keras** es la interfaz de alto nivel de **Tensorflow** desde su versión 1.4.

Otra biblioteca famosa es **Sklearn**, que contiene algoritmos y modelos de aprendizaje automático que se ejecutan con solo inicializarlos. Esta biblioteca no es requerida para este trabajo.

Los códigos fueron llevados a cabo en la plataforma Colaboratory, puesta a disposición totalmente gratuita por Google con acceso a GPUs y TPUs, aunque también oferta una versión profesional del servicio con mayores recursos y menos restricciones. Esta plataforma da acceso a una máquina virtual en la que se ejecuta código de python a través de un cuaderno de Jupyter que se almacena en Google Drive.

Los GPUs disponibles son K80, T4, P4 y P100 de NVIDIA. No se puede elegir entre ellos, son otorgados dependiendo disponibilidad y cantidad de tiempo que el usuario los ha requerido. Lo mismo ocurre con otros recursos como la memoria. El cuaderno puede estar abierto un máximo de 12 horas, aunque, cuando existe una gran demanda, este tiempo puede ser menor. De igual manera pasa con el tiempo que el GPU es utilizable.

3.2. DECODIFICACIÓN DE CÓDIGOS POLARES

La RN trabaja como un clasificador cuando se busca decodificar un mensaje proveniente del canal; se cuenta con un conjunto finito de resultados (palabras de código del diccionario) y la salida de la red es un vector de probabilidades que indica la clase a la que pertenece el mensaje, la clase elegida será aquella con el mayor valor en el vector de salida de la red. Tal vector se genera con la función de activación Softmax.

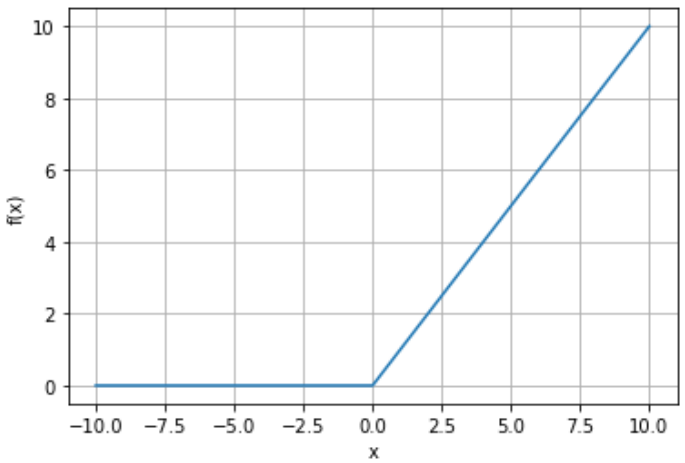
3.2.1. FUNCIÓN DE ACTIVACIÓN.

Una parte central de una RN es la función de activación de sus capas ocultas, es decir, las capas intermedias entre la entrada y la salida. Estas funciones modifican los datos para que la capa subsecuente reciba más información que la que se le presenta en la entrada. Por medio del algoritmo *backpropagation* se ajustan los pesos de cada neurona y con esto la función de activación extraerá

de mejor manera la información requerida para identificar la clase a la que pertenece el mensaje de entrada.

El algoritmo de *backpropagation* realiza derivadas parciales con el objetivo de minimizar el error de cada neurona, el error evidentemente depende de la función de activación.

En [Gruber et al, 2017] se utiliza la función de activación ReLU, Figura 18, como se sugería en la literatura disponible en ese momento [Goodfellow et al, 2017]. Un problema es que esta función manda todos los valores negativos del dominio al valor cero; un problema más grave es que sufre de un punto de discontinuidad en la derivada, esto es un inconveniente cuando se trata de minimizar el error para puntos ubicados en esa región.



$$f(x) = \begin{cases} 0 \rightarrow x < 0 \\ x \rightarrow x > 0 \end{cases}$$

Figura 138. Función ReLU.

Autores han propuesto variaciones que “suavizan” la función ReLU [Clevert et al, 2016], [Klambauer et al, 2017] y así evitan el problema del punto de discontinuidad en la derivada. A continuación se muestra el resultado que se obtiene al usar tales funciones con la topología y optimizador (ADAM) propuestos por [Gruber et al, 2017].

El desempeño que tiene la RN con la función ReLU se muestra en la Figura 19.

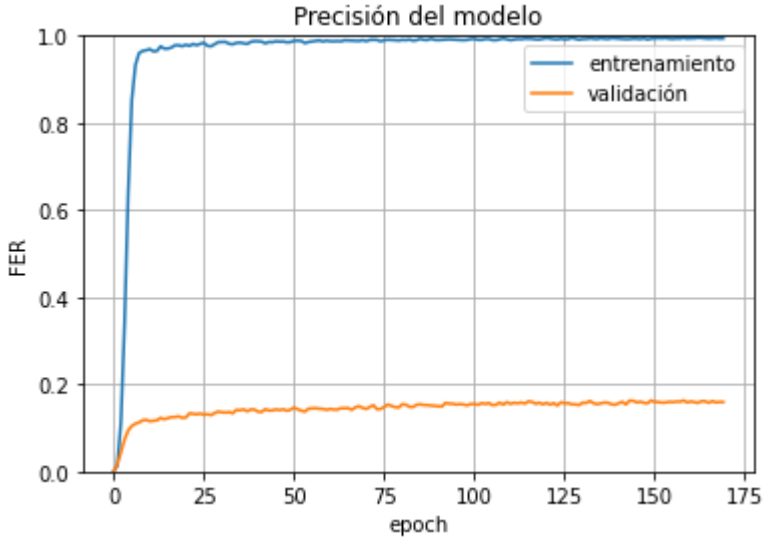


Figura 149. Desempeño de RN con función ReLU.

Para el conjunto de entrenamiento, con menos de 10 *epochs* se llega a una precisión cercana al 100%, mientras que, para el conjunto de validación, la precisión nunca supera el 20%. Esto significa que la red memorizó el conjunto de entrenamiento, por lo tanto, se sufrió de *overfitting.*

La convergencia a la solución depende de la inicialización de los pesos de las neuronas. Si se inician todos con valor cero o uno, en algunos casos la convergencia será lenta, en el caso de la decodificación de códigos polares, jamás se llega a un resultado aceptable.

De manera predeterminada **Tensorflow** utiliza el inicializador Glorot normal; mediante esta técnica las varianzas de los datos de entrada y salida de cada capa (en ambos recorridos de la información) tienden a ser similares si la función de activación es lineal. Dicho efecto conlleva a eliminar gradientes exagerados y con esto evitar un pobre aprendizaje [Glorot et al, 2010]. El inconveniente es que las funciones de activación requieren de la no linealidad para crear fronteras de decisión más complejas.

El método consiste en inicializar los parámetros a entrenar de tal manera que sean normalmente distribuidos con media cero y varianza $\sigma^2 = {}^{2}/_{n_{\text{in}} + n_{\text{out}}}$ en donde n_{in} y n_{out} es el tamaño del vector de entrada y salida de cada capa, respectivamente.

En [Géron, 2020] se propone utilizar el método He, que cuenta con su variante normal y uniforme, en conjunto con la función ReLU. El inicializador fue

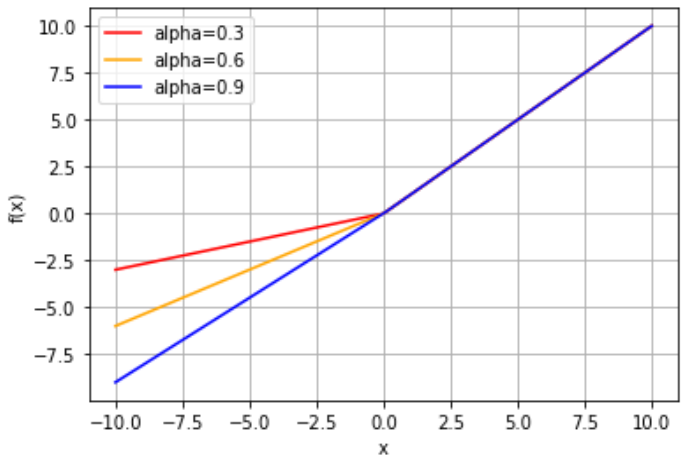
creado con el objetivo de entrenar redes con una mayor cantidad de capas comparado con la literatura de esa época [He et al, 2015]. En su forma normal, la varianza de los pesos debe ser $\sigma^2 = {}^{2}/_{n_{in}}$. En la variante uniforme, el peso debe estar en el rango $\left[-({}^{6}/_{n_{\text{in}}})^{1/2}, ({}^{6}/_{n_{\text{in}}})^{1/2}\right]$. La Tabla 3 contiene el desempeño de cada uno de estos métodos.

Tabla 3. Desempeño de la función ReLU con distintos inicializadores.

ReLU	He normal	He uniforme	Glorot
Prueba	0.1471	0.137	0.1653
Validación	0.15	0.14	0.1678
Entrenamiento	0.9928	0.9947	0.994
Epoch	127	142	186

Para el problema de la decodificación, a pesar de ser una clasificación, el inicializador tipo Glorot consigue mejores resultados en cuanto a precisión en el conjunto de validación.

La primera variante de la función ReLU que se presenta en este texto es la Leaky ReLU, observe la Figura 20. Con esta función aún no se erradica el problema de la discontinuidad en la derivada mencionado con anterioridad, pero se elimina la rigidez en la parte negativa del dominio provocando mejor ajuste en la región de decisión [Maas et al, 2013].



$$f(x) = \begin{cases} \alpha \cdot x \rightarrow x < 0 \\ x \rightarrow x > 0 \end{cases}$$

Figura 20. Función Leaky ReLU.

En la zona en donde los valores se saturan en cero para la función ReLU, todos los gradientes tienen también el valor de cero y esto provoca un lento

aprendizaje de la RN**.** Una vez que el entrenamiento concluyó, es muy probable que la región de decisión construida no sea aceptable. Gracias al parámetro α en la función Leaky ReLU, el problema de constantemente tener gradientes con valor cero se elimina. La Tabla 4 contiene el desempeño con cada método de inicialización y valor de α por defecto en Tensorflow.

Tabla 4. Desempeño de función Leaky ReLU con distintos inicializadores.

Leaky ReLU (0.3)	He normal	He uniforme	Glorot
Prueba	0.3738	0.3479	0.3556
Validación	0.3753	0.3634	0.3569
Entrenamiento	0.9989	0.9922	0.9962
Epoch	146	52	82

Si se compara el mejor desempeño para el conjunto de prueba entre la función ReLU y Leky ReLU, se puede ver un aumento de 2.25 en cuanto a precisión. La Tabla 5 contiene el resultado con distintos valores de α.

Tabla 5. Desempeño de función Leaky ReLU para distintos valores de α.

Leaky ReLU (0.5)	He normal
Prueba	0.4347
Validación	0.455
Entrenamiento	0.9991
Epoch	121

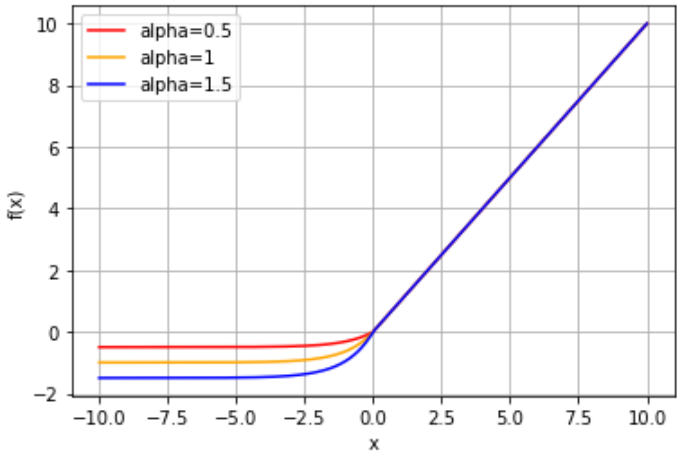
Leaky ReLU (1)	He normal	Glorot
Prueba	0.4558	0.4626
Validación	0.4679	0.4663
Entrenamiento	0.9969	0.9874
Epoch	91	11

Leaky ReLU (1.25)	He normal	Glorot
Prueba	0.4491	0.4314
Validación	0.4553	0.4356
Entrenamiento	0.9962	0.9983
Epoch	87	100

Leaky ReLU (1.5)	He normal	Glorot
Prueba	0.444	0.435
Validación	0.4421	0.4422
Entrenamiento	0.9956	0.9962
Epoch	95	132

Para esta función el mejor resultado se obtiene con α igual a 1 y la inicialización tipo Glorot con un desempeño aproximadamente 2.78 veces mayor al obtenido con la función ReLU. La diferencia más importante se observa en la cantidad de *epochs* que se requieren para el entrenamiento; se necesitan prácticamente 10 veces menos con la función Leaky ReLU.

Una variante suavizada es la función ELU, vea Figura 21. Aquí, el dominio negativo se mapea a una función exponencial eliminando la discontinuidad en la derivada para α=1 y reduciendo la diferencia entre las derivadas del dominio negativo y positivo. El inconveniente es que se llega a una asíntota definida por α, generando el problema de que un conjunto del dominio será saturado al mismo valor.



$$f(x) = \begin{cases} \alpha\left(e^{x} - 1\right) \rightarrow x \leq 0 \\ x \rightarrow x > 0 \end{cases}$$

Figura 21. Función ELU.

Una interesante propiedad de esta función es que los valores de salida de cada capa tienen media cercana a cero, con esto se evita crear una desviación que puede ser perjudicial para el ajuste de la siguiente capa [Clevert et al, 2016]**.** La Tabla 6 contiene el desempeño con cada método de inicialización y el valor de α por defecto, que en este caso es 1.

Tabla 6. Desempeño de función ELU para distintos inicializadores.

ELU (1)	He normal	He uniforme	Glorot
Prueba	0.2344	0.2232	0.2599
Validación	0.2389	0.2249	0.2637
Entrenamiento	0.9929	0.9911	0.9897
Epoch	68	93	93

La mejora es de aproximadamente 1.5 con respecto a la función ReLU. Con la función ELU y con valores de α distintos se genera la Tabla 7.

Tabla 7. Desempeño de función ELU para distintos valores de α.

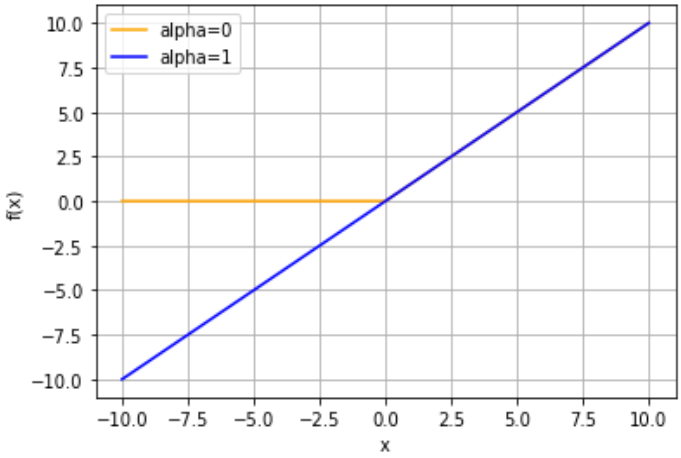
ELU (5)	He normal	Glorot
Prueba	0.221	0.3018
Validación	0.2241	0.3145
Entrenamiento	0.9833	0.9914
Epoch	12	80

ELU (10)	He normal	Glorot
Prueba	0.2088	0.2926
Validación	0.2126	0.2924
Entrenamiento	0.9843	0.9922
Epoch	12	13

Se puede ver que, comparado con la función ReLU, la precisión aumenta casi al doble con este tipo de función.

Dados los resultados anteriores, se observa que el parámetro que identifica a cada función es el responsable de lograr una buena clasificación. La variante de función de activación PReLU contiene un parámetro entrenable que se modifica con el error de clasificación [He et al, 2015]. Al inicio del entrenamiento puede parecer que la función no cambia mucho con respecto a la función ReLU, pero, si se cuenta con una gran cantidad de datos, la función puede ajustarse de manera satisfactoria. En la Figura 22 se observa que la función PReLU es idéntica a Leaky ReLU, solo que el parámetro entrenable va moviendo la pendiente de la recta de la zona negativa del dominio propiciando un mejor ajuste.

El subíndice *i* indica que capa a capa el factor α es distinto**.** Este valor se aprende por medio del algoritmo *backpropagation* y solo se aporta un parámetro más por capa, por lo tanto la complejidad de entrenamiento no crece de manera considerable. La Tabla 8 contiene el desempeño con cada método de inicialización.



$$f(x_i) = \begin{cases} \alpha_i x_i \rightarrow x_i \leq 0 \\ x_i \rightarrow x_i > 0 \end{cases}$$

Figura 22. Función PReLU.

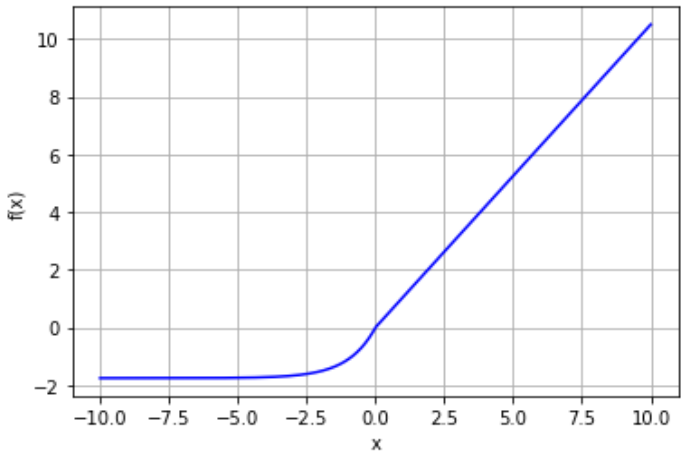
Tabla 8. Desempeño de función PReLU para distintos inicializadores.

PReLU	He normal	He uniforme	Glorot
Prueba	0.1387	0.1923	0.2769
Validación	0.1424	0.1951	0.2759
Entrenamiento	0.9869	0.9895	0.9988
Epoch	154	109	200

La mejora con respecto a la función ReLU es de alrededor de 1.7 en precisión aunque la cantidad de *epochs* que se requieren es mayor, esto debido a los valores de α inadecuados durante el proceso de entrenamiento.

Revisitando el problema del punto de discontinuidad en la derivada en la función ReLU, la variante SELU también es una función suavizada mediante una sección de función exponencial como se muestra en la Figura 23. La diferencia es la salida que arroja cada capa; bajo algunas condiciones se puede mostrar que los valores de salida estarán normalizados con media cero y varianza uno, lo que permite una mayor cantidad de información para las capas subsecuentes [Klambauer et al, 2017].

Para lograr que la RN sea auto normalizable, mediante la función SELU, ésta debe estar compuesta por capas densas, además la entrada a la red debe ser normalizada previamente y es necesario utilizar el método de inicialización de LeCun que es idéntico al de Glorot con la única diferencia de sustituir n_{avg} por n_{in}.



$$f(x) = \alpha_1 \begin{cases} \alpha_0 \left(e^x - 1\right) \rightarrow x \leq 0 \\ x \rightarrow x > 0 \end{cases}$$

$$\alpha_1 > 1$$

Figura 153. Función SELU.

Analizando la función, el factor α_1 se requiere para cumplir los 4 puntos necesarios para una RN auto normalizable establecidos en [Klambauer et al, 2017]: "La función debe tener valores negativos y positivos en el rango para centrar la media en cero. Deben existir regiones de saturación para controlar varianzas grandes en capas finales. La pendiente tiene que ser mayor a 1 para incrementar varianzas pequeñas en capas finales (α_1 es fundamental para este punto) y por último, la curva debe ser continua en todo dominio".

En **Tensorflow** los parámetros están predefinidos en $\alpha_0 = 1.67326324 y \alpha_1 = 1.05070098$. La Tabla 9 contiene el desempeño con cada método de inicialización.

Tabla 9. Desempeño de función SELU para distintos inicializadores.

SELU	LeCun normal	LeCun uniforme	Glorot
Prueba	0.274	0.264	0.2833
Validación	0.2773	0.257	0.2845
Entrenamiento	0.9914	0.9901	0.9936
Epoch	106	82	138

La mejora es de un poco más de 1.7 comparado con la función ReLU. El mayor desempeño se obtiene con un inicializador distinto al de LeCun, indicando que para el problema de la decodificación el inicializador Glorot es la mejor opción; aunque para esta función, se perdería la propiedad de la normalización en el conjunto de datos.

De lo anterior, se tienen 3 candidatos para reemplazar la función ReLU del diseño original tomando en cuenta precisión y cantidad de *epochs*:

1. Leaky ReLU de parámetro 1 e inicializador Glorot normal
2. SELU con inicializadores LeCun normal o Glorot normal
3. ELU de parámetro 5 e inicializador Glorot normal

A pesar de que la función PReLU tiene un desempeño similar a la función SELU, con la cantidad de datos que se tienen se necesita el doble de *epochs* en el entrenamiento, por tal motivo la función queda descartada.

3.2.2. OPTIMIZADOR.

Un optimizador es al algoritmo de *backpropagation* con modificaciones que ayudan a una convergencia más rápida a la solución.

Para fines de comparación, la Tabla 10 muestra los resultados para la topología 128-64-32, función de activación ReLU y distintas tasas de aprendizaje bajo el algoritmo de *backpropagation*.

Tabla 10. Desempeño de *backpropagation*.

Lr	0.1	0.01	0.001
Prueba	0.1186	0.267	0.1909
Validación	0.1208	0.2666	0.1941
Entrenamiento	0.5962	0.9983	1
Epoch	20	34	244

Una modificación al algoritmo de *backpropagation* es incluir un factor llamado *momentum* que toma en cuenta el valor del gradiente de la etapa pasada de entrenamiento con el objetivo de controlar qué tan agresiva (o veloz) debe ser la actualización del parámetro o peso [Rumelhart et al, 1986]**.** La función que define la velocidad es:

$$m_i^{(t)} \leftarrow \beta m_i^{(t-1)} - \lambda \cdot \frac{\partial C}{\partial w_i^{(t)}} \tag{16}$$

En donde el superíndice *(t)* se refiere a la etapa de actualización y β es el factor de *momentum*. La ecuación para actualizar el peso es:

$$w_i^{(t)} \leftarrow w_i^{(t-1)} + m_i^{(t)} \tag{17}$$

Recuerde la ecuación (15), el parámetro se modifica restando la derivada de la función de costo con respecto al parámetro en cuestión. Así, en la etapa inicial $t = 0$ la ecuación (16) solo conserva el valor negativo de la derivada y por la ecuación (17) el resultado es idéntico al *backpropagation* original. En etapas posteriores, el valor de la etapa anterior ayuda en la ecuación (16) a que los descensos hacia el mínimo sean más grandes lo que supone una convergencia más rápida a la solución.

La Tabla 11 muestra los resultados para distintos valores de *momentum* con la configuración de red que se utilizó para la Tabla 10.

Tabla 11. Desempeño de optimizador *momentum*.

Momentum	0.9	0.8	0.7
Prueba	0.1287	0.2456	0.2371
Validación	0.1314	0.2537	0.2425
Entrenamiento	0.5602	0.9995	0.994
Epoch	36	46	44

Este optimizador no resulta conveniente para la tarea de decodificar.

Como segunda modificación se tiene el método de Nesterov que evalúa la función de costo con el peso y la velocidad multiplicada por el factor de *momentum*. De esta manera, cuando se aplica la derivada parcial sobre la función costo, el descenso al mínimo también depende de la dirección y velocidad a la que se apuntó en la etapa pasada de actualización [Sutskever et al, 2013]. La función de velocidad es:

$$m_i^{(t)} \leftarrow \beta m_i^{(t-1)} - \lambda \cdot \frac{\partial C}{\partial\left(w_i^{(t)} + \beta m_i^{(t-1)}\right)} \tag{18}$$

La ecuación de actualización sigue siendo la ecuación (17). La Tabla 12 muestra los resultados para distintos valores de *momentum*.

Tabla 12. Desempeño de optimizador Nesterov.

Nesterov	0.1	0.3	0.5	0.8
Prueba	0.2804	0.2809	0.2623	0.2221
Validación	0.2753	0.2763	0.2597	0.2268
Entrenamiento	0.9962	0.9994	0.9969	0.9817
Epoch	28	56	24	39

Si se compara con el mejor resultado de la Tabla 10 se observa que existe una leve mejora en cuanto a precisión y cantidad de *epochs*, sobre todo para el *momentum* igual a 0.1.

Los optimizadores mencionados anteriormente se enfocan en dirigirse rápidamente al punto que se cree mínimo. En ocasiones no se logra converger a la solución ya que algunas zonas son difíciles de alcanzar (como las situadas entre regiones con mínimos locales) y el gradiente calculado nos puede llevar a un punto totalmente alejado del mínimo global.

El optimizador RMSprop (*Root Mean Square propagation*) funciona reduciendo la tasa de aprendizaje etapa por etapa y así evitar sobresaltos causados por gradientes que se dirigen fuera de la zona del punto mínimo [Hinton, 2012]. La Tabla 13 muestra los resultados para diferentes tasas de aprendizaje.

Tabla 13. Desempeño de optimizador RMSprop.

Tasa aprendizaje	0.01	0.001	0.0001	0.00001
Prueba	0.0001	0.1193	0.3018	0.2342
Validación	0.0001	0.1255	0.3014	0.2359
Entrenamiento	0	0.3199	0.9935	0.8553
Epoch	1	85	75	289

Comparando el mejor resultado de los optimizadores anteriores, se puede observar que el RMSprop es el que otorga mayor precisión a cambio de más *epochs* para lograr tal valor. Esto ocurre por la disminución en la tasa de aprendizaje que hace los pasos más controlados cuando se dirige al mínimo.

El algoritmo RMSprop se muestra a continuación. La ecuación que provoca el cambio de la tasa de aprendizaje es:

$$s_i^{(t)} \leftarrow \rho \cdot s_i^{(t-1)} + (1-\rho)\left(\frac{\partial C}{\partial w_i^{(t)}}\right)^2 \tag{19}$$

La función de actualización es:

$$w_i^{(t)} \leftarrow w_i^{(t-1)} - \lambda \cdot \frac{\partial C}{\partial w_i^{(t)}} \cdot \left(s_i^{(t)} + \varepsilon\right)^{-1/2} \tag{20}$$

El factor al cuadrado en la ecuación (19) sirve para exagerar los gradientes que son grandes, por lo tanto, se logra reducir en mayor medida la tasa de aprendizaje en la ecuación (20). La constante ε es un elemento de seguridad cuando s_i es igual a cero; de cualquier manera, su valor impacta en el resultado final. El factor ρ funciona de la misma manera que el factor β para los optimizadores anteriores. Al igual que con *backpropagation*, es posible incluir *momentum* para acelerar la convergencia.

La Tabla 14 contiene resultados para diferentes valores de ρ, ε y momentum. Los parámetros fijos se mantienen en el valor por defecto de **Tensorflow**.

Se puede ver que la constante ε afecta solamente cuando su valor es alto y que la configuración por defecto en **Tensorflow** con una tasa de aprendizaje de 0.0001 arroja el mejor resultado. Observe que la cantidad de *epochs* es mayor a los optimizadores anteriores por efecto de disminución en el tamaño de los pasos.

Tabla 14. Desempeño de optimizador RMSprop con distintos parámetros.

Rho	0.7	0.8
Prueba	0.0426	0.0715
Validación	0.0428	0.0705
Entrenamiento	0.1003	0.1602
Epoch	230	58

Momentum	0.9	0.5	0.1
Prueba	0.1004	0.028	0.0351
Validación	0.1009	0.0279	0.0352
Entrenamiento	0.1394	0.0246	0.0362
Epoch	17	50	59

Épsilon	1.00E-06	1.00E-08	1.00E-09
Prueba	0.1827	0.3097	0.3039
Validación	0.1879	0.3066	0.3027
Entrenamiento	0.4634	0.9999	1
Epoch	55	78	93

El optimizador propuesto por [Gruber et al, 2017] para lograr la decodificación es el denominado ADAM (*Adaptive Moment Estimation*). De la Tabla 14 se puede concluir que la inclusión de un factor de *momentum* en RMSProp no ayuda a una rápida convergencia a la solución. El optimizador ADAM se basa en aprovechar el control hacia el mínimo, como se hace con RMSProp, mientras incorpora el *momentum* mediante una tasa de decaimiento que lo minimiza en etapas avanzadas de entrenamiento, de tal suerte que se evita dar sobresaltos de gradientes que nos aparten de la región buscada al final del entrenamiento pero se acelera cuando se está lejos el punto mínimo [Reddi et al, 2018]. La Tabla 15 muestra los resultados para diferentes tasas de aprendizaje.

Tabla 15. Desempeño de optimizador ADAM.

Tasa aprendizaje	0.01	0.0001	0.00001
Prueba	0	0.2901	0.1694
Validación	0	0.2901	0.1681
Entrenamiento	0	1	0.999
Epoch	1	137	147

El algoritmo ADAM se muestra a continuación. Primero se calcula la velocidad y la ecuación que provoca la disminución en la tasa de aprendizaje.

$$m_i^{(t)} \leftarrow \beta_1 m_i^{(t-1)} - (1-\beta_1)\cdot \frac{\partial C}{\partial w_i^{(t)}} \tag{21}$$

$$s_i^{(t)} \leftarrow \beta_2 \cdot s_i^{(t-1)} + (1-\beta_2)\left(\frac{\partial C}{\partial w_i^{(t)}}\right)^2 \tag{22}$$

Luego, el control se ejerce mediante las siguientes ecuaciones.

$$\hat{m}_i^{(t)} \leftarrow \frac{m_i^{(t)}}{1-\beta_1^t} \tag{23}$$

$$\hat{s}_i^{(t)} \leftarrow \frac{s_i^{(t)}}{1-\beta_2^t} \tag{24}$$

Los valores de β_1 y β_2 se eligen entre cero y uno. Por último, la función de actualización es:

$$w_i^{(t)} \leftarrow w_i^{(t-1)} - \lambda \cdot \hat{m}_i^{(t)} \cdot \left(\hat{s}_i^{(t)} + \varepsilon\right)^{-1/2} \tag{25}$$

Gracias a las ecuaciones (23) y (24) se desciende de forma cuidadosa, conforme crece la cantidad de iteraciones *t*, el valor del divisor va tendiendo a 1, provocando que el paso sea más pequeño. El valor de ε solamente evita que exista una división entre cero. La Tabla 16 contiene los resultados para distintos valores de β_1, β_2 y ε.

Tabla 16. Desempeño de optimizador ADAM con distintos parámetros.

Beta 1	0.999	0.8	0.5	0.1
Prueba	0.2348	0.2592	0.2684	0.2766
Validación	0.2379	0.2608	0.2673	0.2786
Entrenamiento	1	1	1	1
Epoch	81	142	127	132

Beta 2	0.85	0.8	0.75	0.5
Prueba	0.2948	0.3526	0.3021	0.1158
Validación	0.2948	0.3513	0.3104	0.1185
Entrenamiento	0.9843	0.9999	0.9898	0.2065
Epoch	75	67	36	5

Épsilon	1.00E-08	1.00E-06
Prueba	0.33	0.2993
Validación	0.3328	0.3037
Entrenamiento	0.9789	0.9921
Epoch	83	29

Si se compara el mejor resultado para RMSProp contra el mejor resultado que obtiene el optimizador ADAM, se puede observar que, para el optimizador ADAM, la cantidad de *epochs* disminuye logrando una mayor precisión.

Una variante notable es el optimizador NADAM (*Nesterov Adaptive Moment Estimation*). La modificación consiste en incluir el truco de Nesterov en el algoritmo ADAM, esto debido a que se ha mostrado que para *backpropagation* se obtiene mejor resultado que con *momentum* clásico [Dozat, 2016]. La Tabla 17 muestra los resultados para diferentes tasas de aprendizaje.

Tabla 17. Desempeño de optimizador NADAM.

Tasa aprendizaje	0.001	0.0001	0.01
Prueba	0.1535	0.2733	0
Validación	0.159	0.2771	0
Entrenamiento	0.9334	1	0
Epoch	126	94	1

El algoritmo NADAM se muestra a continuación. Al inicio se calculan las ecuaciones (21) - (24) de igual manera que para el optimizador ADAM. Después se realiza el ajuste de la velocidad por medio de la ecuación (26).

$$\overline{m}_i^{(t)} \leftarrow \beta_1 \hat{m}_i^{(t-1)} + \frac{1-\beta_1}{1-\beta_1^t} \cdot \frac{\partial C}{\partial w_i^{(t)}} \tag{26}$$

Por último, la función de actualización es:

$$w_i^{(t)} \leftarrow w_i^{(t-1)} - \lambda \cdot \overline{m}_i^{(t)} \cdot \left(\widehat{s}_i^{(t)} + \varepsilon\right)^{-1/2} \tag{27}$$

La Tabla 18 contiene los resultados para distintos valores de β_1, β_2 y ε.

Tabla 18. Desempeño de optimizador NADAM con distintos parámetros.

Beta 1	0.999	0.5	0.8
Prueba	0	0.0439	0.0631
Validación	0	0.0446	0.0623
Entrenamiento	0	0.09	0.1344
Epoch	1	50	60

Beta 2	0.75	0.5
Prueba	0.3274	0.1369
Validación	0.3296	0.1364
Entrenamiento	0.9959	0.1726
Epoch	35	6

Épsilon	1.00E-08	1.00E-06
Prueba	0.3284	0.3574
Validación	0.3312	0.3565
Entrenamiento	0.9901	0.9972
Epoch	59	33

De lo anterior, se observa que el optimizador NADAM es el único candidato a reemplazar al optimizador propuesto en [Gruber et al, 2017]. Los restantes son superados tanto en efectividad como en tiempo de convergencia a la solución.

3.2.3. NORMALIZACIÓN Y REGULARIZACIÓN.

De la sección 3.2.2 se identifica al gradiente de la función costo como parte medular del entrenamiento. Aunque los diversos optimizadores fueron creados para buscar el mínimo de manera cautelosa, la estructura de una RN por naturaleza ocasiona que conforme el error se va propagando hacia la capa de entrada los gradientes van tomando valores muy grandes o muy pequeños. Si los valores son grandes, la RN tiende a diverger del mínimo error. Por otra parte, si son pequeños, el ajuste de los pesos será mínimo y el entrenamiento puede ser muy lento o simplemente no se llega a la solución.

Un método para sortear estos problemas es normalizar el *batch* para que este conjunto de datos tenga una media y varianza óptimas. El hecho de tener gradientes inestables es originado por una mala transmisión de información en ambas direcciones a través de la red; dicho defecto puede ser reducido si los datos de entrada y salida de la capa tienen medias y varianzas iguales [Ioffe et al, 2015].

La técnica consiste en calcular la media y varianza de todos los datos del *batch* para normalizar cada vector de entrada, después se agregan dos parámetros a entrenar a cada capa de tal manera que los datos de entrada de la capa tiendan a tener media y varianza adecuadas para el entrenamiento conforme este progresa. Este último paso es necesario debido a que no basta con tener datos de entrada con media cero y varianza uno ya que existe un término incluido por la neurona de desviación o (*bias*) en la capa.

La Tabla 19 muestra los resultados obtenidos aplicando esta técnica a la topología y optimizador de [Gruber et al, 2017].

Tabla 19. Desempeño con normalización de *batch*.

Prueba	0.6437
Validación	0.6377
Entrenamiento	0.9998
Epoch	143

Se puede observar la gran efectividad de este método, con solo modificar los datos de entrada a la capa se logra una precisión aproximadamente 4 veces más grande.

En la sección 3.2.1 se trató acerca de distintas variantes de la función de activación ReLU; una característica importante es eliminar la zona de saturación de dicha función debido a que los gradientes aplicados en esa región son todos igual a cero, provocando malas actualizaciones de parámetros a entrenar. También se habló sobre los distintos inicializadores que tienen por objetivo igualar las varianza de los datos de entrada a cada capa. Junto a la normalización de *batch*, se cuenta con un conjunto de técnicas diseñadas a lograr un aprendizaje más veloz. El inconveniente es que, con tal agresividad en la búsqueda de la solución, se puede sufrir de *overfitting*.

Los regularizadores son una técnica que previene tal falla basados en la idea de que es más difícil que un modelo de RN pequeño memorice los datos de entrenamiento [Chollet, 2018]. La técnica radica en sumar a la función de costo un término de penalización que depende de los parámetros a entrenar. La penalización tiene la función de mantener los pesos en un rango de valores adecuado para forzar al modelo a comportarse como uno más simple a pesar de la cantidad de neuronas que formen la RN. La Tabla 20 muestra los resultados de aplicar la norma ℓ_1.

Tabla 20. Desempeño con norma ℓ_1.

L1 (0.0001)	
Prueba	0.1902
Validación	0.1922
Entrenamiento	0.9909
Epoch	124

La Tabla 21 muestra los resultados de aplicar la norma ℓ_2.

Tabla 21. Desempeño con norma ℓ_2.

L2 (0.001)	
Prueba	0.5199
Validación	0.525
Entrenamiento	1
Epoch	454

L2 (0.0001)	
Prueba	0.1748
Validación	0.1747
Entrenamiento	0.991
Epoch	145

La Tabla 22 muestra los resultados de aplicar la combinación de ambas normas.

Tabla 22. Desempeño con norma ℓ_1 y ℓ_2.

L1(0.0001) L2 (0.001)	
Prueba	0.5113
Validación	0.5128
Entrenamiento	1
Epoch	428

Con esta modificación se alcanza una precisión de aproximadamente 3 veces en los últimos dos casos gracias a que se generaliza de mejor manera aunque la cantidad de *epochs* es el doble de grande debido a la simplificación del modelo.
La Tabla 23 muestra los resultados de aplicar la combinación de usar normalización y regularización.

Tabla 23. Desempeño con normalización de *batch* y norma ℓ_2.

Batch L2 (0.001)	
Prueba	0.6685
Validación	0.6694
Entrenamiento	1
Epoch	449

A pesar de que la mejora no es muy grande, para este tipo de problema se busca precisión, por lo que parece buena idea incluir ambos métodos.

3.2.4. EXPERIMENTOS FINALES.

Para la topología 128-64-32 inicial, se optó por utilizar la función SELU ya que se obtuvieron buenos resultados cuando los datos estaban normalizados, para la capa de entrada se utiliza una normalización de *batch* por ser requerimiento de la función de activación y regularizador ℓ_2 igual a 0.001 en cada capa. El optimizador es ADAM con tasa de aprendizaje igual a 0.0001, β_2 igual a 0.8 y los parámetros restantes con sus valores por defecto. La Tabla 24 muestra los resultados para esta RN.

Tabla 24. Desempeño función SELU y optimizador ADAM.

Prueba	0.7027
Validación	0.7032
Entrenamiento	1
Epoch	129

Se observa que para el set de entrenamiento se llega al 100 por ciento de precisión, se puede concluir que se está sufriendo de *overfitting* a pesar de utilizar el regularizador ℓ_2.

Existe una técnica más agresiva llamada *Dropout* que inhabilita un porcentaje de las neuronas de forma aleatoria en el recorrido de los datos de entrada para forzar a las neuronas restantes a sustraer mayor información por sí solas. Para mantener la propiedad de generar salidas normalizadas de la función SELU, se requiere utilizar el *Dropout* de tipo alpha que conserva media y varianza del vector de datos. Se realizaron pruebas y se encontró que se obtenía el mejor resultado al apagar 2 neuronas únicamente en la primer capa. La Tabla 25 muestra la precisión al aplicar dicha técnica.

Tabla 25. Desempeño con *Dropout*.

Prueba	0.7246
Validación	0.7234
Entrenamiento	1
Epoch	150

Se observa que el *overfitting* sigue causando daños.

En la sección 3.2.1 se observa que la función Leaky ReLU de parámetro 1 sobrepasa todos los resultados, se probó incluirla capa a capa desde la salida hacia atrás para encontrar el mejor funcionamiento. El regularizador ℓ_2 no fue incluido en dichas capas . Los resultados se muestran en la Tabla 26.

Se puede ver que la mayor precisión se obtiene con la primera capa SELU y las restantes Leaky ReLU de parámetro 1. Esto es debido a que la función SELU puede propagar de mejor manera el error por la eliminación del punto de discontinuidad en la derivada; ya que es la primera, es la encargada de encontrar características que las siguientes capas puedan aprovechar. Las siguientes capas

tienen la ventaja de eliminar la saturación de valores en un solo punto y por lo tanto ser más holgada en el momento de ajustarse a los datos.

Tabla 26. Desempeño con combinación de funciones SELU y Leaky ReLU con optimizador ADAM.

Leaky ReLU capa 3	
Prueba	0.7822
Validación	0.7907
Entrenamiento	1
Epoch	205

Leaky ReLU capa 2 y 3	
Prueba	0.8171
Validación	0.8172
Entrenamiento	1
Epoch	246

Leaky ReLU capa 1, 2 y 3	
Prueba	0.8111
Validación	0.8084
Entrenamiento	1
Epoch	141

La siguiente modificación es en el tamaño de la red. Para una topología 256-128-64, función de activación SELU, normalización de *batch* para la primera capa, regularizador ℓ_2 igual a .001 en cada capa y optimizador ADAM igual a la topología anterior, se tienen los resultados de la Tabla 27**.**

Tabla 27. Desempeño con topología 256-128-64.

Prueba	0.8508
Validación	0.8513
Entrenamiento	1
Epoch	90

De nuevo se probó utilizar la función Leaky ReLU de parámetro 1 capa a capa, sin regularizador ℓ_2, y se obtuvo la Tabla 28.

Para este caso ocurre lo mismo que en la primera topología.

Si se usa la función ReLU como en un inicio, con el mismo optimizador se obtienen los resultados de la Tabla 29.

Tabla 28. Desempeño con combinación de funciones SELU y Leaky ReLU con optimizador ADAM y topología 256-128-64.

Leaky ReLU capa 3	
Prueba	0.8633
Validación	0.8608
Entrenamiento	1
Epoch	91

Leaky ReLU capa 2 y 3	
Prueba	0.8722
Validación	8729
Entrenamiento	1
Epoch	74

Leaky ReLU capa 1, 2 y 3	
Prueba	0.8633
Validación	0.8651
Entrenamiento	1
Epoch	76

Tabla 29. Desempeño de función ReLU y topología 256-128-64.

ReLU	
Prueba	0.2653
Validación	0.2718
Entrenamiento	1
Epoch	143

La cual queda muy por debajo de la precisión alcanzada con la combinación de las funciones de activación SELU y Leay ReLU.

Se probó el uso del optimizador NADAM, que resultó una buena variante en la Sección 3.2.2. Con la mejor configuración encontrada en dicha sección para este optimizador, se obtienen resultados superiores que con el optimizador ADAM, y estos son aún más favorables cuando se modifica el factor β_1 a 0.5 como se muestra en la Tabla 30.

Tabla 30. Desempeño de optimizador NADAM y topología 256-128-64.

NADAM	
Prueba	0.8766
Validación	0.8738
Entrenamiento	1
Epoch	75

Beta 1	0.8	0.6	0.5	0.4
Prueba	0.8719	0.8734	0.8791	0.8776
Validación	0.8779	0.877	0.8761	0.8759
Entrenamiento	1	1	1	1
Epoch	79	80	83	76

El problema de *overfitting* sigue presente. Se probó aumentar el desempeño con diferentes configuraciones de *Dropout* sin lograr mejora.

Con la topología 512-256-128 se realizaron pruebas utilizando *Dropout* y la configuración de la RN anterior, los resultados se observan en la Tabla 31.

Tabla 31. Desempeño con distintos valores de *Dropout* y topología 512-256-128.

Dropout	1_2	1_4	1_8
Prueba	0.8823	0.8881	0.8841
Validación	0.8859	0.8939	0.8876
Entrenamiento	1	1	1
Epoch	40	47	48

1_4, 2_2	1_4, 2_4	1_4, 2_2, 3_2	1_4, 2_2, 3_4
0.8909	0.8841	0.8888	0.8819
0.8899	0.8863	0.8905	0.8852
1	1	1	1
43	43	45	42

En donde la señal n_c identifica el número n de neuronas apagadas aleatoriamente en la capa c, por ejemplo, la señal 1_2 significa 2 neuronas apagadas en la capa 1.

Ahora que la red es más grande, el hecho de inhabilitar neuronas no provoca que la estructura sea demasiado simple para clasificar los mensajes de entrada, por tal razón es posible agregar *Dropout* en la segunda capa para aumentar la precisión.

Lo mismo ocurre con la topología 1024-512-256, vea la Tabla 32.

Tabla 32. Desempeño con distintos valores de *Dropout* y topología 1024-512-256.

Dropout	1_2	1_4	1_8
Prueba	0.8859	0.8923	0.8876
Validación	0.892	0.8938	0.8925
Entrenamiento	1	1	1
Epoch	26	28	29

1_4, 2_2	1_8, 2_8	1_16, 2_16
0.8919	0.8941	0.8745
0.8906	0.8895	0.8748
1	1	1
27	27	44

Note que la precisión se encuentra alrededor de 0.89 en las dos estructuras anteriores sin importar que la segunda sea el doble de grande que la primera, la diferencia radica en la cantidad de *epochs* que se requieren para lograr resultados similares. Si se sigue aumentando el tamaño, se observa el mismo comportamiento de disminución en el total de *epochs*, vea la Tabla 33 que corresponde a un tamaño de 2048-1024-512.

Tabla 33. Desempeño con topología 2048-1024-512.

Prueba	0.8916
Validación	0.8895
Entrenamiento	1
Epoch	19

La siguiente etapa es aumentar el número de capas de la RN. Para 4 capas 256-128-64-32, función de activación SELU, normalización de *batch* y regularizador ℓ_2 de 0.001 en la primera capa y Leaky ReLU en las restantes se generan los valores de la Tabla 34.

Tabla 34. Desempeño con topología 256-128-64-32.

4 capas	
Prueba	0.7846
Validación	0.7815
Entrenamiento	1
Epoch	150

Los resultados de la Tabla 35 corresponden a las topologías 512-256-128-64, 1024-512-256-128 y 2048-1024-512-256 respectivamente.

Tabla 35. Desempeño con topologías 512-256-128-64, 1024-512-256-128 y 2048-1024-512-256.

4 capas	1	2	3
Prueba	0.8622	0.8859	0.8913
Validación	0.8691	0.8873	0.8923
Entrenamiento	1	1	1
Epoch	104	43	27

Si se comparan las topologías que tienen la misma cantidad de neuronas en la última capa, la precisión de la RN con 4 capas sin *Dropout* es levemente mejor a la topología de 3 capas con el menor *Dropout*. La cantidad de *epochs* se mantiene pero el tiempo de entrenamiento es casi 10 veces más grande, pasando de aproximadamente 1567 segundos a más de 14000 para 3 y 4 capas respectivamente, esto debido a la gran cantidad de pesos que se agregan con solo una capa más. Lo anterior fue simulado en la plataforma descrita en la sección 3.1.

La calidad en la clasificación realizada por una RN depende en gran medida de la forma en que se le presentan los datos de entrada. Una manera de codificar el mensaje es utilizando la matriz *bit-reversal* en el constructor como se muestra en [Arikan, 2009].

La mejor configuración encontrada para la RN fue usando la normalización de *batch*, función de activación SELU con inicializador LeCun normal y regularizador ℓ_2 de parámetro igual a 0.001 para la primera capa y función de activación Leaky ReLU de parámetro 1 con inicializador Glorot para las restantes, optimizador Nadam con tasa de aprendizaje de 0.0001, parámetros β_1 igual a 0.5 y β_2 igual a 0.999 con ε de 0.00000001. A continuación se muestra la Figura 24 que contiene el desempeño para la topología inicial 128-64-32 con palabras de código generadas con la matriz *bit-reversal* y la configuración de red mencionada anteriormente.

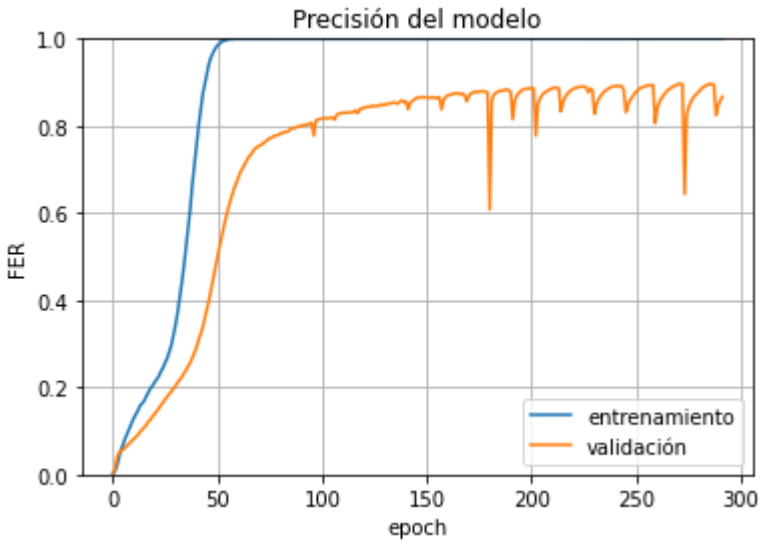


Figura 24. Desempeño de RN con palabras de código generadas con matriz *bit-reversal*.

La precisión para el conjunto de prueba y validación es de alrededor de 0.895. Para el conjunto de entrenamiento la red no tiene fallas. La cantidad de *epochs* para llegar a estos valores fue de un poco más de 250. Comprando con los resultados obtenidos sin usar la matriz *bit-reversal*, la precisión es aproximadamente 0.08 más grande a cambio de alrededor de 30 *epochs* más.

La Tabla 36 contiene el desempeño para distintas topologías y dos neuronas apagadas en la primera capa mediante *Dropout* de tipo alpha.

Tabla 36. Desempeño de RN con palabras de código generadas con matriz *bit-reversal.*

Bit-reversal	128-64-32	256-128-64	512-256-128	1024-512-256
Prueba	0.8961	0.9306	0.9375	0.9339
Validación	0.9007	0.9325	0.9423	0.9404
Entrenamiento	1	1	1	1
Epoch	274	86	47	32

Se puede ver que por fin se rompe la barrera del 90 por ciento de precisión. Como ocurría anteriormente, a mayor topología, menos *epochs* para lograr el entrenamiento. El mejor desempeño se obtiene con la topología 512-256-128, el tiempo de entrenamiento por epoch en promedio fue de 38.32 s, mientras que para la topología 256-128-64 fue de 29.17 s, por lo que esta última puede ser la mejor opción.

Se realizaron pruebas utilizando *Dropout* en las capas siguientes sin encontrar mejora.

Un aspecto primordial es el tiempo de decodificación. Para el decodificador de Cancelación Sucesiva (CS) el tiempo requerido para 20000 palabras de código es de 44.9516 s. Para el decodificador de Cancelación Sucesiva por Lista (CSL) es de 64.9851 s; este decodificador saca ventaja con respecto al anterior cuando el tamaño de la palabra de código es más grande. Con el decodificador *Fast Simplified Successive Cancellation* obtiene una gran disminución en tiempo con 11.8713 s para el mismo conjunto de palabras utilizado con los anteriores decodificadores. Todos los decodificadores alcanzaron una precisión de alrededor de 0.99.

Con ayuda de la programación paralela en GPU y la topología de 256-128-64, el tiempo de decodificación del conjunto de 20000 palabras es de 8.2628 s. Sin GPU el tiempo promedio fue de 21.5851 s, que es considerablemente más

grande que el tiempo anterior pero aún mejor que los decodificadores CS y CSL. Lo anterior fue simulado en la plataforma descrita en la sección 3.1.

CAPÍTULO 4 CONCLUSIONES Y TRABAJO FUTURO

Los resultados obtenidos en este trabajo surgen de modificar la red neuronal completamente conectada de [Gruber et al, 2017] que consiste en una topología 128-64-32 con función de activación ReLU y optimizador ADAM con parámetros no reportados en la literatura.

4.1. CONCLUSIONES.

La principal contribución de este trabajo de investigación consiste en lograr una rápida decodificación de códigos polares mediante la experimentación de una red neuronal completamente conectada. La estructura seleccionada tiene por objeto reducir el tiempo de respuesta asociado al cómputo con respecto a otras redes neuronales que se han probado en la literatura: Convolucional (RNC) y Recurrente (RNR). Además, porque en cierta medida, el proceso de clasificación de una RN puede programarse en paralelo mediante GPUs, reduciendo aún más el tiempo de respuesta. Sin embargo, se identificó que, aunque el tiempo de procesamiento es rápido, la aproximación obtenida no es aceptable en para casos reales de la tecnología 5G.

A continuación se explican los aspectos más relevantes del estudio:

La función de activación ReLU conlleva a sufrir de dos fenómenos fatales para lograr una buena clasificación: la saturación en la región negativa del dominio y el punto de discontinuidad en la derivada para el valor cero, mismos que sus variantes resuelven parcialmente eliminando estos problemas. Dependiendo del uso que se le va a dar a la RN se debe optar por sobrepasar alguno de estos dos obstáculos.

En el problema de la decodificación de códigos polares, se observó que es buena opción eliminar la discontinuidad en la primera capa y así poder penalizar de mejor manera el error que esta tiene, provocando que dicha capa haga una mejor extracción de información para las capas siguientes. Para estas capas subsecuentes, es buena elección eliminar la rigidez que la función ReLU tiene en su dominio negativo; con esto se logran fronteras de decisión con un ajuste más fino y por consiguiente una buena clasificación. Para lograr lo anterior, es fundamental elegir un valor correcto para el parámetro que modifica a estas funciones de activación.

En cuanto al algoritmo de propagación de errores, siempre es buena opción utilizar una variante que permita aumenta y disminuir el tamaño del paso hacia el mínimo de la función de costo dependiendo de lo alejado que se esté de la solución. El inconveniente es que la efectividad depende totalmente de los parámetros con los que va a trabajar tal algoritmo; estos pueden ser encontrados repitiendo el experimento hasta alcanzar un resultado aceptable o por medio de otros algoritmos de validación [Géron, 2020].

Además del mecanismo que utiliza el optimizador para modificar los pesos, se observó que el control de estos mediante un inicializador, normalizador y regularizador otorga buenos dividendos debido a que la estructura de la RN también está inmiscuida en el entrenamiento y no depende en su totalidad del algoritmo de propagación del error.

Utilizando la estructura de una RN completamente conectada para la decodificación de códigos polares, se sufre de *overfitting*; el método de *Dropout* ayuda si solo se aplica a la primera capa y se apagan únicamente dos neuronas, debido a que esta capa es más grande y no se simplifica en gran medida la topología de la red. Un *Dropout* más agresivo es beneficioso con topologías de 512 y 1024 neuronas en la primera capa; por el tamaño de estas dos redes, se obtienen mejoras usando *Dropout* en las capas subsecuentes.

La mejor topología encontrada fue 256-128-64. Para la primera capa, la función de activación es la SELU, que es una versión sin punto de discontinuidad de la clásica función ReLU. Para el correcto funcionamiento de la función SELU, se requiere normalizar el conjunto de datos de entrada e iniciar los parámetros a entrenar con el método de LeCun. También se incluyó el regularizador ℓ_2 de parámetro 0.001 que otorgó una gran mejora en el desempeño. Para la segunda y tercera capa se utilizo la función de activación Leaky ReLU de parámetro igual a 1, que es una versión de la función ReLU que elimina su zona de saturación de valores que se encuentra en el rango negativo de la función. El mejor inicializador de parámetros a entrenar en las últimas dos capas fue el de Glorot en su versión normal. Dado que la red, para la tarea de decodificador funciona como un clasificador de múltiples clases, la capa de salida tiene la función Softmax como función de activación.

El método de propagación de errores elegido es el *Nesterov Adaptive Moment Estimation*, que puede acelerar y desacelerar el descenso hacía el mínimo error a conveniencia y mostró ligeramente mejores para topologías más grandes comparado con el método ADAM. Los parámetros de funcionamiento de este método fueron: tasa de aprendizaje igual a 0.0001, β_1 igual a 0.5, β_2 igual a 0.999 y ε igual a 10^{-8}.

Con respecto al tiempo de decodificación, la RN derrota a los decodificadores por Cancelación Sucesiva, Cancelación Sucesiva por Lista y *Fast Simplified Successive Cancellation* cuando se realiza programación paralela en un GPU en la plataforma de simulación descrita en la sección 3.1. En el tema de la precisión, estos decodificadores alcanzan el 99% de efectividad mientras que la RN logra alrededor del 93%, por lo que aún no es posible implementarla en un sistema real con tecnología 5G.

4.2. TRABAJO FUTURO.

Como continuación de este trabajo de tesis se presenta la siguiente lista, sin orden de importancia, que contiene temas sobresalientes que, por el alcance del trabajo no fueron tratados.

1. Probar la efectividad de la RN propuesta en este trabajo en canales *Non-AWGN* y presentar una comparación con los métodos de decodificación probabilísticos. Esto con el objetivo de probar la estructura de red neuronal completamente conectada en aplicaciones reales.
2. Aplicar variantes del método *K-fold cross-validation* para la elección de parámetros con el objetivo de mejorar el desempeño de la RN. Dado que la configuración de red en este trabajo se realizó de la manera prueba-error, es posible encontrar una mejor configuración que ofrezca una precisión aceptable.
3. Modificar los parámetros de las arquitecturas de las redes convolucionales y recurrentes presentadas en [Lyu et al, 2018] como se hizo con la arquitectura de [Gruber et al, 2017]. Así como ocurrió con la estructura de red completamente conectada, con parámetros correctos es posible aumentar el desempeño en cuanto a tiempo de cómputo y precisión que las redes convolucionales y recurrentes ofrecen.
4. Aumentar el tamaño de la palabra de código incluyendo la RN encontrada en este trabajo en la arquitectura propuesta en [Doan et al, 2018], basada en

dividir la palabra de código en sub bloques que sirven para entrenar varias redes que en conjunto decodifican el mensaje completo. En este trabajo se utilizaron palabras de código que quedan muy por debajo del requerimiento de palabras de 512 bits en escenarios reales.

5. Tratar de mejorar el decodificador propuesto en [Nachmani et al, 2018] que utiliza una red neuronal recurrente para entrenar parámetros que son usados en la gráfica de factores del decodificador *Belief Propagation*. Con lo anterior, se puede identificar si las redes neuronales son mejor opción como decodificadores o, por otra parte, auxiliares de decodificadores probabilísticos.

6. Incluir el optimizador MIND de [Jiang et al, 2019] en la RN propuesta en este trabajo. Como se observó en este trabajo, la elección del método de optimización marca una gran diferencia en cuanto a la precisión obtenida. El optimizador MIND fue diseñado con el objetivo de mejorar decodificadores con redes neuronales y es posible que la precisión de la estructura de red completamente conectada aumente con su uso.

7. Probar nuevas arquitecturas de red como la denominada "*Transformer*" [Vaswani et al, 2017]. Dado que aún no se ha publicado un decodificador que utilice esta estructura, es importante investigar su desempeño para la tarea de decodificar códigos polares.

La dificultad de implementar otro tipo de estructuras a la red neuronal completamente conectada radica en el el poder de cómputo necesario para lograrlo; recordar que la red neuronal completamente conectada es la más sencilla. Lo mismo ocurre con los métodos de validación para encontrar los parámetros de la red, se requiere de una considerable cantidad de iteraciones que consisten en entrenar la red y evaluar su desempeño.

REFERENCIAS BIBLIOGRÁFICAS

- [3GPP, 2011]. 3GPP, “Physical layer, Measurements,” TS-36.214,V10.1.0, 2011.

- [3GPP, 2020]. 3GPP, “Study on New Radio (NR) access technology,” TR-38.912, V16.0.0, 2020.

- [3GPP, 2017]. 3GPP, “Study on New Radio Access Technology, Physical Layer Aspects ,” TR-38.802, V14.2.0, 2017.

- [3GPP, 2019]. 3GPP, “Technical Specification Group Services and System Aspects,” TR-21.915, V15.0.0, 2019.

- [Alpaydin, 2010]. E. Alpaydin, *Introduction to machine learning 2° ed.*, The MIT Press, 2010.

- [Andrews et al, 2014]. J.G. Andrews, S. Buzzi, W. Choi, S. Hanly, A. Lozano, A. Soong & J.C. Zhang, “What Will 5G Be?,” *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065-1082, June 2014, doi: 10.1109/JSAC.2014.2328098.

- [Arikan, 2009]. E. Arikan, “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels,” *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051-3073, July 2009, doi: 10.1109/TIT.2009.2021379.

- [Bennatan et al, 2018]. A. Bennatan, Y. Choukroun and P. Kisilev, "Deep Learning for Decoding of Linear Codes - A Syndrome-Based Approach," 2018 IEEE International Symposium on Information Theory (ISIT), 2018, pp. 1595-1599, doi: 10.1109/ISIT.2018.8437530.

- [Bergstra et al, 2011]. J. S. Bergstra, R. Bardenet, Y. Bengio, & B. Kégl, “Algorithms for hyper-parameter optimization,” *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2011.

- [Buzzi et al, 2016]. S. Buzzi, C. I, T. E. Klein, H. V. Poor, C. Yang & A. Zappone, “A Survey of Energy-Efficient Techniques for 5G Networks and Challenges Ahead,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 4, pp. 697-709, April 2016, doi: 10.1109/JSAC.2016.2550338.

- [Cammerer et al, 2017]. S. Cammerer, T. Gruber, J. Hoydis & S. Brink, "Scaling Deep Learning-Based Decoding of Polar Codes via Partitioning," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1-6, 2017, doi: 10.1109/GLOCOM.2017.8254811.

- [Chollet, 2018]. F. Chollet, *Deep learning with python*, Manning, 2018.

- [Cover et al, 2006]. T. M. Cover & J. A. Thomas, *Elements of information theory 2° ed*., Wiley-Interscience, 2006.

- [Clevert, 2016]. D. Clevert, T. Unterthiner, S. Hochreiter, “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),” *ICLR*, 2016.

- [Dimnik et al, 2009]. I. Dimnik & Y. Be’ery, “Improved random redundant iterative hdpc decoding,” *IEEE Transactions on Communications*, vol. 57, no. 7, pp. 1982–1985, 2009.

- [Doan, et al, 2018]. N. Doan, S. Ali Hashemi & W. J. Gross, "Neural Successive Cancellation Decoding of Polar Codes," *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1-5, 2018, doi: 10.1109/SPAWC.2018.8445986.

- [Doan et al, 2019]. N. Doan, S. A. Hashemi, E. N. Mambou, T. Tonnellier & W. J. Gross, “Neural Belief Propagation Decoding of CRC-Polar Concatenated

Codes," *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1-6, 2019, doi: 10.1109/ICC.2019.8761399.

- [Doan et al, 2020]. N. Doan, S. A. Hashemi & W. J. Gross, "Decoding Polar Codes with Reinforcement Learning," *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1-6, 2020, doi: 10.1109/GLOBECOM42002. 2020.9348007.

- [Dozat, 2015]. T. Dozat, "Incorporating Nesterov Momentum into," 2015.

- [Elkelesh et al, 2018]. A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "Belief propagation decoding of polar codes on permuted factor graphs," *IEEE Wireless Commun. and Net. Conf.*, pp. 1–6, Apr. 2018.

- [Fan et al, 2016]. Y. Fan, C. Xia, J. Chen, "A Low-Latency List Successive-Cancellation Decoding Implementation for Polar Codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 303-317, Feb. 2016, doi: 10.1109/JSAC.2015.2504318.

- [Gallager, 1962]. R.G. Gallager, "Low-density parity-check codes." *IRE Trans. Inf. Theory 8*, pp. 21-28, 1962.

- [Gallager, 1968]. R. G. Gallager. *Information theory and reliable communication*, John Wiley & Sons,1968.

- [Géron, 2020]. A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow 2° ed.*, O'Reilly, 202.

- [Giard et al, 2017]. P, Giard, C. Thibeault, W. J. Gross. *High-speed decoders for polar codes*, Springer, 2017.

- [Goodfellow et al, 2017]. I. Goodfellow, Y. Bengio, & A. Courville, Deep learning: The MIT Press, pp. 800, 2017 ISBN: 0262035618. Genetic Programming and Evolvable Machines. 19. 10.1007/s10710-017-9314-z.

- [Glorot et al, 2010]. X. Glorot & Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” *Journal of Machine Learning Research - Proceedings Track 9*. pp.249-256, 2010.

- [Gruber, et al,2017]. T. Gruber, S. Cammerer, J. Hoydis & S. T. Brink, “On deep learning-based channel decoding,” *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, 2017, pp. 1-6, doi: 10.1109/CISS.2017.7926071.

- [Hashemi et al, 2017]. S. A. Hashemi, C. Condo & W. J. Gross, “Fast Simplified Successive-Cancellation List Decoding of Polar Codes,” *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 1-6, 2017.

- [Hastie et al, 2008]. T. Hastie, R. Tibshirani, J. Friedman, *The elements of statistical learning: data mining, inference, and prediction 2° ed.*, Springer, 2008.

-[He et al, 2015]. K. He, X. Zhang, S. Ren & J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026-1034, 2015, doi: 10.1109/ICCV. 2015.123.

- [Hebb, 1949]. D.O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, Wiley & Sons, 1949.

- [Hornik et al, 1989]. K. Hornik, M. Stinchcombe, H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359-366, 1989, ISSN 0893-6080, https://doi.org/10.1016/0893-6080(89)90020-8.

- [Hui et al, 2018]. D. Hui, S. Sandberg, Y. Blankenship, M. Andersson & L. Grosjean, "Channel Coding in 5G New Radio: A Tutorial Overview and

Performance Comparison with 4G LTE", *IEEE Vehicular Technology Magazine*, vol. 13, no. 4, pp. 60-69, Dec. 2018, doi: 10.1109/MVT.2018.2867640.

- [Ioffe et al, 2015]. S. Ioffe & C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, vol. 37, pp. 448–456, 2015.

- [Jiang et al, 2019]. Y. Jiang, H. Kim, H. Asnani & S. Kannan, "MIND: Model Independent Neural Decoder," *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1-5, 2019, doi: 10.1109/SPAWC.2019.8815537.

- [Jiang et al, 2019b]. Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh & P. Viswanath, "Turbo Autoencoder: Deep learning based channel codes for point-to-point communication channels," 2019.

- [Kim et al, 2018]. H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh & P. Viswanath, "Communication Algorithms via Deep Learning," *ArXiv*, *abs/1805.09317*, 2018.

- [Kingma et al, 2014]. D. P. Kingma & J. L. Ba, "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.698*0. Dec. 2014 .

- [Klambauer et al, 2017]. G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, "Self-Normalizing Neural Networks," 2017.

- [Liao et al, 2020]. Y. Liao, S. A. Hashemi, J. Cioffi and A. Goldsmith, "Construction of Polar Codes with Reinforcement Learning," *IEEE Transactions on Communications*, 2020, doi: 10.1109/TCOMM.2021.3120274.

- [Lin et al, 1983]. Shu Lin, Daniel J. Costello *Error control coding: Fundamentals and applications*, Prentice-Hall, 1983.

- [Lugosch et al, 2017]. L. Lugosch & W.J. Gross. Neural, “Offffset min-sum decoding”. *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 1361–1365, June 2017. ISSN 2157-8117. doi: 10.1109/ISIT.2017.8006751.

- [Lutz et al, 2003]. M. Lutz, D. Ascher, *Learning Python 2° ed.*, Oreilly & Associates Incorporated, 2003.

- [Luo, 2020]. F. Luo. *Machine Learning for Future Wireless Communications*, Wiley, 2020.

- [Lyu et al, 2018]. W. Lyu, Z. Zhang, C. Jiao, K. Qin & H. Zhang, "Performance Evaluation of Channel Decoding with Deep Neural Networks," *2018 IEEE International Conference on Communications (ICC)*, pp. 1-6, 2018, doi: 10.1109/ICC.2018.8422289.

- [Maas et al, 2013]. A. L. Maas, A. Y. Hannun, A. Y. Ng, “Rectifier Nonlinearities Improve Neural Network Acoustic Models,” 2013.

- [McCulloch et al, 1943]. W.S. McCulloch & W. Pitts, “A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5,” pp. 115–133, 1943. https://doi.org/10.1007/BF02478259

- [Minsky, 1959]. M. L. Minsky, “Some Methods of artificial intelligence and heuristic programming,” *Mechanization of Thought Processe*s, vol. 1,1959.

- [Nachmani et al, 2016]. E. Nachmani, Y. Be'ery & D. Burshtein, "Learning to decode linear codes using deep learning", *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2016, pp. 341-346, doi: 10.1109/ALLERTON.2016.7852251.

- [Nachmani et al, 2018]. E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein and Y. Be’ery, "Deep Learning Methods for Improved Decoding of Linear Codes", *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119-131, Feb. 2018, doi: 10.1109/JSTSP.2017.2788405.

- [O´Shea et al, 2017]. T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563-575, Dec. 2017, doi: 10.1109/TCCN. 2017.2758370.

- [Popovski et al, 2018]. P. Popovski, K. F. Trillingsgaard, O. Simeone and G. Durisi, "5G Wireless Network Slicing for eMBB, URLLC, and mMTC: A Communication-Theoretic View," *IEEE Access*, vol. 6, pp. 55765-55779, 2018, doi: 10.1109/ACCESS.2018.2872781.

- [Reddi et al, 2018]. S. J. Reddi, S. Kale & S. Kumar, "On the convergence of Adam & Beyond," *ICLR,* 2018.

- [Ren et al, 2015]. Y. Ren, C. Zhang, X. Liu, and X. You, "Effificient early termination schemes for belief-propagation decoding of polar codes," *IEEE 11th Int. Conf. on ASIC*, pp. 1–4, Nov 2015.

- [Rosenblatt, 1963]. F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," *American Journal of Psychology 76*, 1963.

- [Rumelhart et al, 1986]. D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning representations by back-propagating errors." *Nature 323*, pp. 533-536, 1986.

- [Russell et al, 1995]. S. J. Russell & P. Norvig, *Artificial intelligence a modern approach*, Prentice Hall, 1995.

- [Sarkis et al, 2016]. G. Sarkis, P. Giard, A. Vardy, C. Thibeault and W. J. Gross, "Fast List Decoders for Polar Codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 318-328, Feb. 2016, doi: 10.1109/JSAC.2015.2504299.

- [Sayood, 2012]. K. Sayood, *Introduction to Data Compression 4° ed*., Elsevier, 2012.

- [Shannon, 1948]. C. E. Shannon, “A Mathematical Theory of Communication,” *The Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656*, Oct. 1948.

- [Sutskever et al, 2013]. I. Sutskever, J. Martens, G. Dahl & G. Hinton, “On the importance of initialization and momentum in deep learning.” *In Proceedings of the 30th International Conference on International Conference on Machine Learning*, vol. 28 (ICML'13), JMLR.org, III–1139–III–1147, 2013..

- [Sutton et al, 2018]. R. S. Sutton & A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA, USA: A Bradford Book, 2018.

- [Tahir et al, 2017]. B. Tahir, S. Schwarz & M. Rupp, "BER comparison between Convolutional, Turbo, LDPC, and Polar codes," 2017 24th International Conference on Telecommunications (ICT), 2017, pp. 1-7, doi: 10.1109/ICT.2017.7998249.

- [Tal et al, 2012]. I. Tal & A. Vardy, "List Decoding of Polar Codes," IEEE Transactions on Information Theory, vol. 61, no. 5, pp. 2213-2226, May 2015, doi: 10.1109/TIT.2015.2410251.

- [Tal et al, 2013]. I. Tal & A. Vardy, "How to Construct Polar Codes," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562-6582, Oct. 2013, doi: 10.1109/TIT.2013.2272694.

- [Tariq et al, 2019]. F. Tariq, M. R. A. Khandaker, K. -K. Wong, M. A. Imran, M. Bennis & M. Debbah, "A Speculative Study on 6G", *IEEE Wireless Communications*, vol. 27, no. 4, pp. 118-125, Aug. 2020, doi: 10.1109/MWC.001.1900488.

- [Turing, 1937]. A.M. Turing, “On Computable Numbers, with an Application to the Entscheidungs-problem,” *Proceedings of the London Mathematical Society*, s2-42: 230-265, 1937, https://doi.org/10.1112/plms/s2-42.1.230.

- [Vaswani et al, 2017]. A. Vaswani N. Shazeer, N. Parmar, J. Uszkoreit L, Jones, A. Gomez, L. Kaiser & I. Polosukhin, “Attention Is All You Need,” 2017.

- [VonNeumann, 1958]. J. VonNeumann, *The Computer and the Brain*, Yale University Press, 1958.

- [Xu et al, 2017]. W. Xu, Z. Wu, Y. Ueng, X. You & C. Zhang, "Improved polar decoder based on deep learning", *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2017, pp. 1-6, doi: 10.1109/SiPS.2017.8109997.

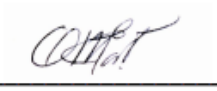
- [Yuan et al, 2014]. B. Yuan and K. K. Parhi, “Early stopping criteria for energy-effificient low-latency belief-propagation polar code decoders”, *IEEE Trans. Signal Process*, vol. 62, no. 24, pp. 6496–6506, Dec 2014.

Firma de aprobación del Jurado de Tesis:

Dr. Giselle Monserrat Galván Tejada,
Departamento de Ingeniería Eléctrica
Sección de Comunicaciones

Dr. Manuel Mauricio Lara Barrón,
Departamento de Ingeniería Eléctrica
Sección de Comunicaciones

Dra. Ana María Antonia Martínez Enríquez,
Departamento de Computación



Dr. Amilcar Meneses Viveros,
Departamento de Computación