Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias
en Ciencias de la computación

Confiabilidad y rendimiento adaptativo de un almacenamiento seguro en la multi nube con redes neuronales

Tesis
para cubrir parcialmente los requisitos necesarios para obtener el grado de
Maestro en Ciencias

Presenta:

Israel Rescalvo Anastacio

Ensenada, Baja California, México
2021

Tesis defendida por

Israel Rescalvo Anastacio

y aprobada por el siguiente Comité

Dr. Andrey Chernykh
Director de tesis

Dr. Edgar Chávez Gonzáles

Dr. Raúl Rivera Rodríguez

Dr. José Antonio García Macías

Dr. Pedro Gilberto López Mariscal
Coordinador del Posgrado en Ciencias de la Computación

Dr. Pedro Negrete Regagnon
Director de Estudios de Posgrado

Resumen de la tesis que presenta **Israel Rescalvo Anastacio** como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Confiabilidad y rendimiento adaptativo de un almacenamiento en la multi nube con redes neuronales

Resumen aprobado por:

_____

Dr. Andrey Chernykh
Director de tesis

El uso de los servicios en la nube se ha incrementado en los últimos años por lo que usuarios, industrias y gobiernos depositan grandes conjuntos de datos en la infraestructura de los proveedores de servicios (CSP, por sus siglas en inglés). No obstante, algunos servicios como el almacenamiento como servicio involucran severos riesgos de accesibilidad, integridad y privacidad de datos. Una solución a dichos inconvenientes está dada por el uso de múltiples CSPs, evitando así que un único CSP cuente con acceso total a la información confidencial. En este trabajo de tesis, se propone el análisis, diseño e implementación de una red apuntadora para configurar un sistema redundante de números residuales con aproximación de rango (AR-RRNS, por sus siglas en inglés) que permite distribuir la información en n CSPs y recuperar la misma solo con k de ellos. La red neuronal propuesta busca minimizar la probabilidad de pérdida de información y redundancia, ambos objetivos están en conflicto y es fundamental determinar una adecuada configuración de (k, n) donde $2 \leq k \leq n$. Asimismo, la red permite seleccionar CSPs específicos y asignarles un segmento del total de la información. La red apuntadora utiliza un modelo codificador-decodificador y un sistema de atención entrenados mediante aprendizaje por refuerzo. Esta estructura le permite a la red afrontar cambios en los parámetros de los CSPs como la probabilidad de error. Además, la mayoría de los cálculos se realizan de manera offline. A partir de los resultados, se llevó a cabo un análisis comparativo con diversas versiones de algoritmos genéticos y el algoritmo de ramificación y poda, todos ellos forman parte fundamental del estado del arte en la optimización. El análisis muestra que la red apuntadora es más eficiente en tiempo que los demás algoritmos y genera soluciones similares en calidad al algoritmo genético simple pero la diversidad es menor con respecto a los algoritmos basado en población.

Palabras clave: almacenamiento en la nube, optimización multi objetivo, Sistema Numérico de Residuo, seguridad, esquemas de compartición de secretos.

Abstract of the thesis presented by **Israel Rescalvo Anastacio** as a partial requirement to obtain the Master of Science degree in Computer Science

Reliability and Adaptive performance of multi-cloud storage with neural networks

Abstract approved by:

_____

Dr. Andrey Chernykh
Thesis Director

The use of cloud services has increased in recent years as users, industries and governments deposit large data sets in the infrastructure of service providers (CSP). However, some services such as storage as a service involve severe data accessibility, integrity and privacy risks. A solution to these drawbacks is given by the use of multiple CSPs, thus preventing a single CSP from having full access to confidential information. In this thesis work, the analysis, design and implementation of a pointing network is proposed to configure a redundant system of residual numbers with range approximation (AR-RRNS) that allows to distribute the information in n CSPs and recover the same only with k of them. The proposed neural network seeks to minimize the probability of information loss and redundancy, both objectives are in conflict and it is essential to determine an adequate configuration of $2 \leq k \leq n$. Likewise, the network allows selecting specific CSPs and assigning them a segment of the total information. The pointer network uses an encoder-decoder model and attention system trained by reinforcement learning. This structure allows the network to face changes in the parameters of the CSPs such as the probability of error. Also, most of the calculations are done offline. Based on the results, a comparative analysis was carried out with different versions of genetic algorithms and the branching and bound algorithm, all of which are a fundamental part of the state of the art in optimization. The analysis shows that the pointing network is more efficient in time than the other algorithms and generates solutions similar in quality to the simple genetic algorithm, but the diversity is lower with respect to population-based algorithms.

Keywords: Multi-objective optimization, cloud storage, Residue Number System, security, Secret Sharing Schemes

Dedicatoria

A mi madre.

Agradecimientos

Al Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California (CICESE). Gracias por permitirme conocer la ciencia en México.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme apoyo económico para mis estudios de maestría. No. de becario 19264566.

A mi asesor de tesis, el Dr. Andrey Chernykh, por permitirme desarrollar este trabajo de investigación.

Los miembros del comité de tesis, el Dr. Edgar Leonel Chávez, el Dr. Raúl Rivera Rodríguez y el Dr. José Antonio García Macías.

Gracias a mis compañeros y al equipo de trabajo del laboratorio en especial al Dr. Jorge Mario Cortés Mendoza.

Notación

n	Cantidad de piezas en la que se divide la información en el sistema AR-RRNS.
k	Cantidad mínima de piezas para poder recuperar la información en el sistema AR-RRNS.
n_s	Valor inferido por la Ptr-Net para el parámetro n.
k_s	Valor inferido por la Ptr-Net para el parámetro k.
N	Conjunto de nubes $N = \{c_1, c_2, c_3, \dots, c_N\}$.
$Pr(k,n)\ o\ Pr$	Probabilidad de pérdida de la información en AR-RRNS.
Pr_{nl}	$Pr_{ln} = ln(Pr)$.
Pr_{n_ln}	Valor Pr_{nl} normalizado.
err_j	Probabilidad de falla en la nube c_j.
R	Redundancia de la información en el sistema AR-RRNS.
R_n	Redundancia normalizada.
D	Tamaño de la información original.
D_E	Tamaño de la información después de ser codificada.
X	Secuencia de entrada $X = \{err_1, err_2, \dots, err_N, \alpha, \beta\}$ para la Ptr-net.
$\pi^{(v)}$	Secuencia solución inferida por la Ptr-Net, $\pi^{(v)} = \{c_2, c_5, c_3, \dots c_n, \alpha, \beta\}$.
C_s	Conjunto solución de nubes encontrada por la Ptr-Net, subsecuencia de $\pi^{(v)}$.
α	Elemento artificial para determinar k_s.
β	Elemento artificial para definir el fin de una secuencia valida $\pi^{(v)}$.
WS	Valor que representa la suma ponderada de los objetivos.
w_x	Peso de la preferencia del usuario por el objetivo x, utilizado para obtener WS.
$\nabla_{variable}$	Gradiente de la función con respecto a la variable.
$p(\ a\|b)$	Probabilidad del evento a dado el evento b.
$p_\theta(\pi^{(v)}\|N)$	Probabilidad de obtener la secuencia $\pi^{(v)}$dado el conjunto N bajo una política parametrizada por θ.

$J(\theta \mid N)$	Aproximación estocástica del desempeño esperado de la Ptr-Net.
$\{enc_i\}_{i=1}^{n}$	Vector columna de memoria latente del codificador, cada enc_i es un elemento codificado de dimensión d del elemento $x_i \in X$.
$\{dec_i\}_{i=1}^{n}$	Vector columna de memoria latente del decodificador, cada dec_i es un elemento codificado de dimensión d.
$A(ref, q_i; W_{ref}, W_q, v)$	Sistema apuntador que forma parte de la Ptr-Net.
ref	Matriz de referencia de dimensión $n \times d$ conformada por los elementos.
q_i	Elemento de consulta del mecanismo apuntador en la iteración i.
W_{nombre}	Matriz cuadrada de dimensión $d \times d$ para parametrizar mecanismos de atención.
$\boldsymbol{v}$	Vector columna de dimensión d para parametrizar mecanismos de atención.
u_j^i	Valor que representa el nivel de atención del elemento i en la iteración j.
$\pi(j)$	j-ésimo elemento inferido por la Ptr-Net.
g_l	Elemento generado con el mecanismo de atención para reemplazar al argumento q_i en el mecanismo apuntador.

Tabla de contenido

Lista de figuras

Lista de tablas

Capítulo 1 Introducción

El continuo desarrollo de tecnologías emergentes ha permitido la reducción de costes e impulsado la aparición de grandes centros de datos remotos, los cuales representaron el punto de inflexión para la aparición del cómputo en la nube (L. Qian et al., 2009). La adopción masiva de este paradigma de cómputo moderno se debe principalmente a la alineación de sus beneficios y modelo de negocio con las necesidades inherentes de la industria.

El cómputo en la nube se caracteriza por delegar las complicaciones técnicas relacionadas con el mantenimiento de la infraestructura a un tercero remoto. Siendo así, no se requiere un manejo directo por parte del usuario para acceder a recursos computacionales de almacenamiento o procesamiento de datos. Entre la amplia gama de ventajas ofrecida por el cómputo en la nube destaca su alta escalabilidad, elasticidad, disponibilidad y eficiencia. Además, el usuario solo paga por lo que utiliza al ser un servicio bajo demanda (Armbrust et al., 2010).

El Instituto Nacional de Estándares y Tecnología (NIST por sus siglas en inglés) definió en 2011 tres principales modelos para la nube: Software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS) (Mell y Grance, 2011). Dicha taxonomía ha sido ampliada en años recientes dada las crecientes necesidades de los usuarios, incluyendo servicios tales como almacenamiento como servicio (STaaS), comunicaciones como servicio (CaaS), red como servicio (NaaS), monitoreo como servicio (MaaS), aprendizaje de maquina como servicio (MLaaS), entre muchos otros (Kachele et al., 2013).

En particular, STaaS nace a partir de la masiva utilización de los servicios en la nube para almacenar y compartir información. En STaaS, un usuario convencional almacena un conjunto de datos para su resguardo en un servidor remoto, donde gran parte del proceso es transparente para él. Sin embargo, el recurrir a un proveedor de servicios en la nube (CSP, por sus siglas en inglés) implica potenciales riesgos de privacidad, integridad y disponibilidad de los datos (Cachin et al., 2009).

Si bien los CSPs cuentan con protocolos y mecanismos internos para el aseguramiento y resguardo de los datos, estas técnicas no garantizan en lo absoluto que la seguridad sea llevada a cabo de manera adecuada. Incluso, no se garantiza el aviso oportuno a los afectados sobre algún fallo o robo de datos (Marston et al., 2011). Siendo así, a fin de mitigar las diversas vulnerabilidades de estos servicios, se han propuesto los

modelos multinube donde la información sensible se almacena en dos o más CSPs para evitar que un único CSP cuente con acceso total a los datos.

Este trabajo se enfoca en el Sistema Redundante de Números Residuales con Aproximación de Rango (AR-RRNS, por sus siglas en inglés) propuesto por Chervyakov et al. (2019), este sistema permiten distribuir la información entre distintos CSPs y resolver varias problemáticas inherentes a los sistema distribuidos.

El proceso efectuado por un sistema basado en AR-RRNS consiste en: Primero, establecer la cantidad de pedazos a dividir la información (n) y la cantidad mínima de pedazos necesarios para recuperar esta información (k), donde $2 \leq k \leq n$. Luego, la información es tratada de acuerdo con la configuración del sistema elegida y cada pedazo se entrega a un CSP distinto. Finalmente, el proceso de recuperación consiste en descargar al menos k pedazos y reconstruir la información con base en el teorema del residuo chino (CRT, por sus siglas en inglés).

La elección de los parámetros $(k,\ n)$ y los CSPs tienen un impacto significativo en el desempeño del sistema. Bajo esta premisa, en esta investigación se explora el desarrollo de un sistema complementario basado en redes neuronales artificiales (NN, por sus siglas en inglés), la red apuntadora (Ptr-Net) puede responder a cambios dinámicos en los parámetros de los CSPs involucrados. La Ptr-Net es entrenada utilizando técnicas de Aprendizaje por Refuerzo (RL, por sus siglas en inglés) para la selección de las nubes y los valores (k, n). Este documento presenta los detalles de la implementación, una comparativa del desempeño de Ptr-Net con respecto a otras estrategias empleadas en la literatura y los hallazgos más importantes en cuanto a su calidad y eficiencia.

1.1 Antecedentes

El almacenamiento de datos en servidores de terceros, como en el modelo de STaaS, trae preocupaciones inherentes con respecto a la privacidad, disponibilidad e integridad; lo que es entendible desde la perspectiva de los clientes. Diversas investigaciones han tratado de abordar estas problemáticas para buscar soluciones que aprovechen las ventajas del entorno de la nube y minimicen sus desventajas. Estas consideraciones son expuestas en la primera parte de esta sección. Posteriormente, se abordan los antecedentes de los métodos explorados para configurar el sistema basado en el Sistema Numérico Residual (RNS, por sus siglas en inglés). Finalmente, se analiza el estado del arte en el dominio de redes neuronales para la resolución de problemas combinatorios.

1.1.1 Compartición de secretos en la nube

Compartir un secreto entre múltiples participantes ha sido una línea de investigación relevante en las últimas décadas dados los beneficios que un sistema de esta índole ofrece a los sistemas distribuidos de almacenamiento de información (Tchernykh et al., 2019). Los métodos tradicionales de encriptación cumplen la función de proteger la información, pero no suelen adaptarse a las necesidades que surgen en los sistemas distribuidos bajo distintos escenarios.

El fragmentar un secreto entre múltiples participantes es una alternativa que ofrece seguridad a costa de confiabilidad o conveniencia. Con respecto a la confiabilidad, la información se divide entre diferentes locaciones logrando que si una de ellas no está disponible no se pierda el secreto. Con relación a la conveniencia, al no haber consentimiento por parte de los participantes estos pueden paralizar las actividades del grupo (Shamir, 1979). Un solo participante pudiese tener la información completa o bien, todos podrían tener una copia de la misma (por conveniencia); no obstante, resulta más seguro la existencia de un consenso mínimo de participantes para poder acceder a ella.

En un esquema de compartición de secretos (SSS, Secret Sharing Schema) no es necesario que todos los participantes (n) estén presentes, solo se necesita un subconjunto mínimo requerido (k) de ellos para recuperar el secreto compartido. El sistema umbral (k, n) es un nombre ampliamente adoptado para referirse a este tipo de sistemas.

Shamir (1979) propuso un método SSS basado en interpolación de polinomios que requiere un alto consumo de memoria. Subsecuentemente, con base en las criticas expuestas por Blakley (1979) sobre los métodos de interpolación y su facilidad de inferir información de ellos, Shamir implementó el uso de aritmética modular para dar frente a estas vulnerabilidades.

Blakley (1979) generó un SSS funcional con aritmética modular y el uso de una matriz cuadrada con números del campo de los enteros $\mathbb{Z}$ módulo p. El sistema selecciona una submatriz que permitirá recuperar la información. Una desventaja importante de este enfoque es la necesidad de probar varias combinaciones hasta que se cumpla una condición dada.

Mignotte (1979) propuso el sistema Mignotte-SSS donde utilizó algebra modular para dividir el secreto, junto con condiciones propuestas que habilitan al sistema como un sistema umbral (k, n) y CRT como elemento principal para recuperar el secreto.

Asmuth y Bloom (1983) propusieron el sistema Asmuth-Bloom SSS que usa CRT para recuperar la información. Al debilitar una de las condiciones de primalidad establecidas para el sistema se puede identificar sí algún pedazo de información ha sido deliberadamente manipulada, aunque podría haber una colusión para lograr esto último.

Rabin (1989) propuso el sistema IDA (del inglés, *Information Dispersal Algorithm*) con importantes similitudes a los previos SSS. IDA cuenta con un umbral (k, n) para recuperar la información. Además, se puede obtener información parcial del mensaje con tan solo $k - 1$ pedazos, pero no reconstruirlo. Por último, IDA es eficiente en espacio comparado con Asmuth-Bloom-SSS, Shamir-SSS y Blakley-SSS.

Krawczyk (1994) propuso utilizar un método capaz de mantener la seguridad a un "nivel computacional", es decir, únicamente un atacante con recursos teóricamente infinitos puede vulnerar la seguridad de dicho sistema. Este sistema genera una llave criptográfica C aleatoriamente. Posteriormente, la llave se utiliza para encriptar el secreto S con un algoritmo criptográfico simétrico. Finalmente, la información encriptada S es dividida en n pedazos mediante IDA. De igual manera, C se divide con Shamir-SSS en n pedazos $C_{i \leq n}$ y es esparcida junto con un pedazo $S_{i \leq i}$, tal que cada participante tendrá una tupla (C_i, S_i).

Por otra parte, debido a la rápida adopción de los servicios de la nube y sus derivados, han surgido muchos retos y riesgos anteriormente no contemplados (Zissis y Lekkas, 2012). Donde métodos tradicionales si bien cumplen su cometido, estos no cuentan con el potencial para afrontar tales panoramas, como es el caso del TTP (del inglés, *Trusted Trust Party)* cuando no se adapta para los cambios pertinentes (Marium et al., 2012).

Bessani et al., (2013) presentaron el sistema DepSky para esparcir la información a través de múltiples nubes comerciales usando distintos métodos de seguridad, este sistema se centra en resolver varios problemas y retos del entorno de la nube mediante la aplicación de diferentes tecnologías. Entre sus beneficios se encuentra la confidencialidad, integridad y disponibilidad de la información. Los métodos de seguridad empleados son: un sistema criptográfico de clave pública RSA (Rivest, Shamir y Adleman) con llaves de 1024 bits para firmas, SHA-1 (Secure Hash Algorithm) para hashes criptográficos, AES (del inglés, Advanced Encryption Standard) para encriptación simétrica, el esquema PVSS (del inglés, *Publicly Verifiable Secret Sharing*) de 192 bits para la compartición de secretos y Reed-Solomon para la corrección de errores. Cabe mencionar que el compartimiento de secretos se realiza como lo especificado por Krawczyk (1994).

Con el creciente manejo de datos de múltiples fuentes surgió la necesidad de la administración de documentos altamente sensibles como la historia clínica electrónica por parte de los centros de salud. Ermakova y Fabian (2013) propusieron como medida adicional el uso de SSS para reducir los errores de encriptación o el compromiso de las llaves para desencriptar por parte de los CSPs. Además, el sistema permite compartir los archivos entre diferentes centros, aumentar la disponibilidad y recuperar la información a pesar de que una parte de la información almacenada haya sido manipulada.

Rathanam y Sumalatha (2014) crearon un sistema que utiliza un esquema de almacenamiento flexible y eficiente para asegurar la accesibilidad de los datos y su exactitud en la nube. El sistema consta de una versión mejorada del RSA con doble encriptación para el acceso, encriptación de la información y la técnica Huffman para la compresión de los datos. Por último, los autores hacen uso de un mecanismo auditor de almacenamiento distribuido para superar las limitaciones de la perdida de datos. Desafortunadamente, no se cubre el problema inherente al uso de un único CSP.

Pundkar y Shekokar (2016) implementaron un sistema funcional que permite subir archivos multimedia a tres CSPs con el uso de Shamir-SSS, aunque el tamaño de memoria requerida por el sistema puede ser extremadamente grande considerando que los datos multimedia utilizan un gran volumen y el método Shamir-SSS utiliza el mismo espacio en memoria que la información original por cada fragmento originado.

En ese mismo año, Babitha y Babu (2016) implementaron un sistema para dividir la información y encriptar cada parte individualmente con el algoritmo AES de 128 bits. La seguridad del sistema recae en la autenticación del usuario y en la encriptación de los distintos pedazos en la cual fue dividida la información. Además, como medida adicional, cada pedazo de información tiene un número de identificación que solo es conocido por el usuario. Sin embargo, todos los pedazos de información son almacenados en un solo CSP en diferentes locaciones de memoria.

Celesti et al. (2016) proponen un sistema multinube que proporciona disponibilidad a largo plazo, ofuscación y encriptación. El algoritmo comprime la información para después dividirla con RRNS y generar pedazos redundantes. Posteriormente, cada pedazo de información es codificado en BASE64, encriptado con AES-256 y envuelto con XML para ser enviado a diferentes CSPs. Las llaves criptográficas simétricas y la ubicación de los pedazos son resguardadas por el usuario final.

Chervyakov et al. (2019) desarrollan un sistema basado en RRNS con propiedades similares a Mignotte-SSS con propiedades de detección y corrección de errores. Los autores introducen el concepto de

aproximación de rango (AR, siglas en inglés), como una adaptación de CRT para reducir la complejidad computacional al momento de recuperar la información.

Miranda-López et al. (2018) realizaron una comparación entre Mignotte-SSS y Asmuth-Bloom-SSS con distintas configuraciones del umbral (k, n). Los autores concluyen que si bien Asmuth-Bloom- SSS es un sistema perfecto e idealmente asintótico, el tiempo de codificación y decodificación es ocho veces mayor que Mignotte-SSS de acuerdo a las simulaciones realizadas.

Tchernykh et al. (2018a) propusieron un sistema RNNS anti colusión basado en versiones modificadas de Asmuth-Bloom-SSS y Mignotte-SSS. El sistema evita colusiones entre los participantes que resguardan pedazos de la información encriptada y colaboran para recuperarla sin autorización. Los autores realizaron un análisis teórico para demostrar la seguridad computacional del sistema. Tchernykh et al. (2018b) presentaron un sistema ponderado RRNS con pedazos de menor tamaño y en mayor cantidad; las pedazos son asignados a distintos CSPs de acuerdo con su confiabilidad y respetando ciertas consideraciones para evitar que el CSP más confiable obtenga tantos pedazos que le permita recuperar la información por sí solo, *ergo,* una clara violación *per se*.

1.2 Optimización con redes neuronales

Modelos de NNs están comenzando a ser utilizadas para la resolución a problemas combinatorios. Por ejemplo, mediante arquitecturas como las redes de Hopfield (Hopfield y Tank, 1985) y redes auto organizadas (Kohonen, 1982). Sin embargo, estas presentan problemas referentes a la validez de las soluciones encontradas, incluso con las soluciones subóptimas.

Los múltiples avances en el área y el reciente ánimo de la comunidad científica con la llegada de la tercera ola de las NNs provocaron el desarrollo de soluciones y herramientas para afrontar problemas combinatorios mediante nuevos enfoques.

Los primeros avances propusieron soluciones para la traducción de textos usando secuencia-a-secuencia (Sutskever et al., 2014) y Neuro máquinas de Turing (Graves et al., 2014). Estas arquitecturas fueron desarrolladas para resolver distintos problemas combinatorios. Sin embargo, estos trabajos no lograban superar las dificultades que surgen de tener clases estáticas en la salida, ya que en diversos problemas

combinatorios las clases dependen de la entrada, en otras palabras, se debe tener clases en salida de la red que varíen en cada iteración t de la NN.

Vinyals et al. (2015) identificaron este problema y propusieron una red apuntadora (Ptr-Net) para superar este impedimento, esta nueva NN arquitectura usa la estructura codificador-decodificador e integra un mecanismo de atención para solucionar el problema. Estos mecanismos permiten variar la salida de la red (clases) con respecto a la entrada en cada iteración. Esta NN es entrenada con aprendizaje supervisado para resolver problemas como: casco convexo, triangulación Delaunay y el agente viajero.

El principal reto en el entrenamiento supervisado de una NN para resolver problemas combinatorios yace en los datos de entrenamiento. Generalmente, una heurística genera esta información pero la naturaleza de los datos no garantiza que su generación sea adecuada para el entrenamiento. Bello et al. (2016) proponen el uso del aprendizaje por refuerzo para el entrenamiento de estas NNs, los autores demuestran la mejora en el desempeño que una red de este tipo puede alcanzar al no estar acotada por los datos de entrenamiento obtenidos de un tercer algoritmo. Esto permite a la NN desarrollar su propia heurística para resolver los problemas que se le plantean y evitar imitar el comportamiento de otro procedimiento.

Esta clase de algoritmos basados en NNs permiten desarrollar soluciones para problemas del mundo real, por ejemplo: Yu et al. (2019) propone un sistema para resolver el problema de enrutamiento de vehículos en línea (“online”) mejor conocido como VRP (de inglés, *vehicule routing problem*). La NN mejora el desempeño de una flota de vehículos de entrega en tiempo real (on-line) con condiciones dinámicas donde los pedidos de los vehículos cambian cada cierto tiempo y se debe decidir que pedidos recoger, entregar y definir los tiempos de recarga de los vehículos eléctricos. Otras aplicaciones en el dominio de NNs para optimización combinatoria son: robótica para la recolección de pelotas (Gu et al., 2018), resolver rutas de drones (Khoufi et al., 2019) y el problema de empaquetamiento 3-D (Hu et al., 2017).

La ventaja de utilizar NNs con aprendizaje por reforzamiento es transferir gran parte del cómputo a un enfoque de entrenamiento offline para problemas combinatorios y así ser aplicados en escenarios on-line. Donde otros algoritmos al mínimo cambio del sistema tiene que volver a realizar gran parte del procesamiento desde un inicio (Smith, 1999).

1.3 Objetivos

1.3.1 Objetivo general

Diseñar y analizar una red apuntadora para la optimización de la confiabilidad y redundancia de un sistema de almacenamiento multinube configurable basado en AR-RRNS, la red debe encontrar la configuración (k, n) donde n define el número de pedazos a dividir la información y k establece el número de pedazos necesarios para recupera la información; $2 \leq k \leq n$.

1.3.2 Objetivos específicos

- Analizar el estado del arte en sistemas de almacenamiento distribuido y redes neuronales.
- Diseñar e implementar la red neuronal apuntadora utilizando el framework PyTorch.
- Establecer las configuraciones de experimentación para proveedores reales de nube.
- Evaluar y comparar la eficiencia de la red neuronal apuntadora con algoritmos utilizados en trabajos previos. Para encontrar deficiencias y beneficios al utilizar esta clase se sistemas en conjunto con sistemas de tipo umbral.

Capítulo 2 Cómputo en la nube y seguridad de almacenamiento

La idea detrás del paradigma de la nube no es nueva, desde los años 60 se proponía un sistema de utilidad que fue introducido formalmente hasta el 2006 (L. Qian et al., 2009). La popularización de la computadora personal (pc) en los años 80 hizo perder impulso a esta idea, ya que el usuario final contaba con la capacidad computacional conveniente en una pc, esta idea predominó por mucho tiempo (Vaquero et al., 2008). Sin embargo, la reducción de costes y la mejora de muchas tecnologías hicieron posible la introducción de la nube en la práctica; este concepto inicialmente provocó una confusión generalizada acerca de su definición (Armbrust et al., 2010).

La introducción del paradigma del cómputo en la nube, o simplemente referido como la nube, fue paulatino, la mayoría de sus conceptos y tecnologías fueron desarrolladas años atrás, diversos autores la consideraban simplemente una palabra de moda donde la mercadotecnia es su mayor atributo (Vaquero et al., 2008).

2.1 La nube computacional

Varios trabajos en el área, por ejemplo Vaquero et al. (2008), comparan distintas definiciones y características para intentar definir el concepto de la nube, además los autores explican la diferencia de la nube con otros paradigmas con similitudes como las “grids”.

El paradigma de la nube computacional dejo de ser un simple concepto en los años 00, ya que compañías tecnológicas como Amazon, Google y Microsoft comenzaron a desplegar servicios de este tipo a gran escala desde el 2006 (L. Qian et al., 2009). La masificación de estos servicios provoco que el NIST estableciera una definición para disipar muchas dudas y formalizar el computo en nube; la definición ofrecida por Mell y Grance (2011) establece que:

El computo en la nube es un modelo para habilitar el acceso a una piscina de recursos informáticos compartidos y configurables (p. ej. redes, servidores, almacenamiento, aplicaciones y servicios); de manera ubicua, conveniente y bajo demanda, que pueden ser rápidamente provistos y liberados con el mínimo esfuerzo de gestión o contacto con el proveedor de servicios.

Esta definición también incluye cinco características esenciales, tres modelos de servicios y cuatro modelos de despliegue.

Las cinco características son las siguientes:

- Auto servicio bajo demanda,
- Amplio acceso de red,
- Piscina de recursos,
- Rápida elasticidad,
- Medición del servicio.

Los tres modelos de servicio son:

- **Software como servicio** (SaaS, del inglés Software as a Service): La capacidad provista al usuario es poder utilizar aplicaciones del proveedor que se encuentran ejecutando sobre la infraestructura de la nube.
- **Plataforma como servicio** (PaaS, del inglés Platform as a Service): La capacidad provista al usuario es poder desplegar sus propias aplicaciones o de terceros sobre la infraestructura de la nube mediante librerías, lenguajes, servicios y herramientas soportadas por el proveedor. El usuario solo tiene control sobre sus aplicaciones y tal vez configuraciones del ambiente del hospedaje donde se encuentre la aplicación.
- **Infraestructura como servicio** (IaaS, del inglés Infrastructure as a Service): La capacidad provista al usuario consiste en recursos informáticos, como lo es la capacidad de procesamiento, almacenaje, redes y otros recursos básicos, donde puede desplegar y ejecutar software arbitrario.

Los cuatro modelos de despliegue están definidos por:

- **Nube privada**: La infraestructura en la nube es para uso exclusivo de una sola organización con múltiples clientes. Puede ser poseído, administrado y operado por la organización o un tercero y puede estar dentro o fuera de las instalaciones. Este modelo de despliegue es criticado por algunos autores por su parentesco con un centro de datos tradicional en ciertas circunstancias (Zhang et. al., 2010).

- **Nube comunitaria**: La infraestructura es para el uso exclusivo de una comunidad de clientes que pertenecen a organizaciones que tienen preocupaciones en común. Puede ser operada por diversas organizaciones en la comunidad, un tercero o alguna combinación de ellos, puede existir dentro o fuera de las instalaciones de la organización/s.

- **Nube pública**: Esta infraestructura es provisionada para su uso por parte de un público general. En general la propiedad, instalaciones, la administración, etc. es llevado por el CSP.

- **Nube hibrida**: Es la unión de dos o más modelos de despliegue que permanecen como entidades únicas, pero trabajan de forma conjunta por medio de software propietario o estandarizado.

Los modelos de servicios definidos por el NIST son de carácter muy general, la aparición de nuevos servicios requiere de una taxonomía más fina. Por ejemplo, el almacenamiento como servicio (STaaS, del inglés Storage as a Service) considera los centros datos y los sistemas de archivos, este tipo de servicio se enfoca en ofrecer una manera de resguardar datos de manera persistente (Kachele et al., 2013).

Cuando se habla de seguridad en la nube se consideran tres aspectos principales privacidad, disponibilidad e integridad (AlZain et al., 2012; Zissis y Lekkas, 2012):

- Integridad: Los datos solo deben ser modificados por usuarios con autorización o de maneras autorizadas. En otras palabras, evitar que los datos sean borrados, modificados o fabricados.

- Disponibilidad: Los datos se encuentran en un sistema accesible y usable bajo demanda por un usuario autenticado.

- Privacidad: Controlar la divulgación de información personal.

Algunos de estos problemas pueden darse por errores humanos, mala configuración, personal no confiable, amenazas ambientales, ataques informáticos, etc. La multi tenencia puede ocasionar que un ataque dirigido a otra entidad afecte a otros usuarios del proveedor atacado(Tchernykh et al., 2019; Zissis y Lekkas, 2012).

Recientemente, la alianza de seguridad en la nube publicó los incidentes de seguridad más destacados que han tomado lugar en la nube en los últimos años (2020), algunos de ellos son:

- Un ex ingeniero de Amazon AWS con conocimiento interno en vulnerabilidades ganó acceso y robó documentos sensibles de carpetas protegidos en la nube (2019);

- Un proveedor externo de Down Jones no protegió con contraseña documentos confidenciales hospedados en AWS *Elastic Search*, los documentos eran confidenciales y con consecuencias legales (2017);

- Un proveedor de servicios externo de Tecso almacenó fotografías sensibles en la nube sin ningún sistema de autenticación (2019);

- Un atacante ganó acceso a kubernetes inseguros propiedad de Tesla donde se manejaba propiedad intelectual de los autos de prueba y posiblemente secretos de comerciales. Además, instalo software malicioso para minar criptomonedas (2019);

- Entre otros sucesos que incluyen, robo de información, ataques de denegación de servicio, robo de credenciales, etc.

Estos eventos crean desconfianza y evitan que empresas y usuarios alojen sus datos en la nube, mayormente porque la información está fuera del alcance del usuario. AlZain et al. (2012) enfatizan "*La información es administrada y compartida en servidores percibidos como no seguros y de desconfianza*".

Una solución para abordar los retos planteados es utilizar el concepto de la multinube.

2.2 La multinube

Un STaaS conformado por una sola nube tiene muchos problemas para ser un modelo confiable en el almacenamiento de datos, el uso de diferentes nubes es una manera de mitigar las desventajas de este modelo. De esta forma, se tiene una segunda capa en la pila de servicios donde se utilizan diferentes protocolos en la capa que se encuentra por debajo (Armbrust et al., 2010; Cachin et al., 2009; Vukolić, 2010).

El modelo de la multinube fue descrito como esencial por Vukolić (2010), "Es esencial distribuir la seguridad, confiabilidad y disponibilidad a través de distintos proveedores de nube para mejorar la oferta que cualquiera tendría por sí solo".

Un único CSP presenta un solo punto de fallo, el cual es el CSP mismo. El bug en un solo proveedor puede afectar sus servidores en distintas locaciones o las copias de respaldo de la información almacenada. Al contar con distintos CSPs, se tiene la posibilidad de múltiples: geolocalizaciones de las bases de datos, software, protocolos, arquitecturas, fuentes de energía, hardware, etc. Esta situación no podría ser posible o viable económicamente de afrontar por un CSP (Vukolić, 2010).

Así también, se evita quedar atrapado con un solo proveedor (en inglés, *vendor lock-in*) al distribuir la información entre múltiples proveedores de forma que ninguno de ellos sea dominante sobres los demás, ya que los costos de mover parte de la información son menores y afrontables para una compañía (AlZain et al., 2012).

2.3 Seguridad

2.3.1 Shamir Secret Share Scheme (Shamir-SSS)

EL algoritmo Shamir-SSS (1979) se basa en un principio matemático básico, dadas k tuplas de valores $\{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$ existe una y solo una función $f(x)$ de grado $k-1$ tal que $f(x_i) = f(y_i)$ para toda i.

Tomando lo anterior en cuenta, se puede crear un sistema umbral (k, n) de manera que la k es la cantidad mínima de puntos necesarios para poder aproximar el polinomio $f(x)$, el cual es quien mantiene el secreto en un coeficiente. El proceso para compartir un secreto es el siguiente.

Sea un secreto $s \in \mathrm{Z}$ y un polinomio $f(x)$ de grado $k-1$ talque $f(x) = a_0 + a_1 x^1 + \cdots + a_{k-1} x^{k-1}$ donde a_0 es el secreto s. Entonces, cada pedazo s_i es obtenida al evaluar el polinomio en n puntos distintos, sujeto a $n \geq k$. Con lo cual obtenemos un conjunto de n pedazos $s_1, s_2, \ldots, s_n$.

Algunas técnicas de interpolación de polinomios se utilizan para recupera la información, normalmente interpolación de Taylor. Entonces, con k puntos e interpolación de polinomios se puede aproximar el valor de $f(0)$, este valor define el secreto original s. Nótese que este es el punto donde el polinomio seleccionado corta el eje de las ordenadas en un plano $2D$.

Shamir-SSS es considerado uno de los sistemas más seguros pero de gran consumo de memoria, ya que cada pedazo tiene al menos $b = log_2(s)$ bits de tamaño, que corresponde al tamaño original del secreto, por consiguiente, el espacio en memoria es equivalente al realizar n copias del secreto S.

2.3.2 Mignotte Secret Share Scheme (Mignotte-SSS)

El esquema presentado por Mignotte (1979) utiliza n números primos o coprimos definido por el conjunto $\{m_1, m_2, \ldots m_n\}$. La condición que habilita al sistema como umbral (k, n) esta dada por: sea $a, b \in \mathbb{Z}^+$ y $0 < a < b$, entonces

$$\prod_{i+i}^{k} m_i > b > a > \prod_{i=n-k+1}^{n} m_i \qquad (1)$$

El secreto s a proteger debe ser un número entero tal que $a \leq s \leq b$. Los pedazos son los residuos de los respectivos n módulos seleccionados, de esta manera se tiene que $s_i = s(mod\, m_i)$ para el conjunto $\{s_1, s_2, \ldots, s_n\}$ con sus respectivos módulos $\{m_i, m_2, \ldots, m_n\}$.

La seguridad de este sistema reside en que tan grande sea la diferencia de a y b. Esta consideración puede ser frágil sobre todo en definiciones rigorosas como a la propuesta por Quisquater et al. (2002) pero muy

útil en sistemas donde se priorice preservar espacio en memoria sobre seguridad. Para recuperar el mensaje se utiliza CRT, un subconjunto con al menos k pedazos y sus respectivos módulos.

2.3.3 Asmuth Bloom Secrete Share Scheme (Asmuth-Bloom-SSS)

Presentado por Asmuth y Bloom (1983), este sistema usa $n+1$ primos o coprimos a pares para conformar un conjunto de módulos $\{m_0 < m_1, m_2, \dots m_n\}$ donde m_0 puede ser tomado o no como la llave secreta del sistema. El sistema tiene que cumplir la siguiente condición:

$$\prod_{i=1}^{k} m_i > m_0 \prod_{i=1}^{k-1} m_{n-i+1} \tag{2}$$

El secreto es un entero $s \in [0, m_0)$ con $y = s + A \cdot m_0$ donde A es un entero elegido aleatoriamente y sujeto a la condición $0 \leq y < P = \prod_{i=1}^{k} m_i$. Los pedazos s_i están dadas por $y_i \equiv y(mod\, m_i)$ con $0 \leq y_i < m_i$. Al conocer al menos k pedazos y sus respectivos módulos es una condición suficiente para recuperar y utilizando el CRT. Una vez que se conoce y, se puede obtener s en consecuencia, tal que se obtiene el residuo de y con respecto a m_0. Este esquema es considerado seguro porque el mensaje original se ofusca con la suma de m_0 escalado con un número aleatorio.

2.3.4 Advanced Encryption Standard (AES)

El AES es un sistema de encriptación simétrico que sustituto al DES como resultado de un concurso llevado a cabo por el NIST al buscar un sistema realmente seguro, el nombre original del algoritmo es Rijndael pero oficialmente conocido como AES (Daemen y Rijmen, 2002), ambos nombres son utilizados indistintamente hoy en día para referirse al algoritmo.

Este es un método de encriptación por bloques, es decir, el algoritmo encripta porciones de bits de una misma longitud, resultando en otra porción con la misma longitud que la original.

El método toma dos entradas para realizar la encriptación: el texto plano y una llave de longitud 128, 192 o 256 bits. El método realiza una operación inicial, para posteriormente entrar en un ciclo donde se realizan la misma serie de operaciones una y otra vez, a cada iteración del ciclo se le denomina ronda. El

algoritmo realiza una ronda final ligeramente diferente a las rondas del ciclo. La cantidad de rondas a efectuar dependerá de la cantidad de bits de la llave y el mensaje donde el número mínimo de rondas es 10 y máximo es 14.

El algoritmo utiliza una serie de tablas llamadas cajas de sustitución (s-box) preestablecidas por los autores del algoritmo, las cuales son utilizadas como tablas de consulta para reemplazar las secuencias originales. Las cajas de sustitución son elaboradas de una manera especial que permite fortalecer la encriptación. Las operaciones como sumas y multiplicaciones se efectúan en un campo finito (campo de Galois).

2.3.5 Sistema Numérico de Residuo (RNS)

El sistema numérico de residuo (RNS, siglas en inglés) presentado por Garner (1959) es un sistema finito que tiene distintas ventajas que otros sistemas numéricos no cuentan. Principalmente, es un sistema que no depende de la posición de sus elementos para determinar su valor, en otras palabras, es un sistema no posicional.

Además, el sistema cuenta con propiedades homomórficas con las cuales se determina un parentesco entre dos grupos. Esto es, entre cualesquiera dos elementos pertenecientes al grupo se puede realizar una operación definida para el grupo; tal que habrá una operación y elementos equivalentes en el segundo grupo. Estas características lo vuelven un candidato idóneo para sistemas distribuidos de almacenamiento, debido a que las operaciones se pueden realizar de manera paralela entre los diversos participantes que tienen posesión de la información (Chervyakov et al., 2019).

Otra característica importante es la eficiencia de las operaciones definidas para el grupo debido a que las operaciones aritméticas no usan acarreo entre elementos a diferencia de un sistema posicional convencional. Pero, se debe tener en cuenta el rango que el sistema permite representar para que las operaciones sean válidas y no se presente un desbordamiento (“overflow”), lo que significa que el resultado no puede ser representado por el sistema definido (Mohan, 2016). No todos los problemas son candidatos para ser resueltos con estos sistemas, solo un subconjunto de ellos.

El sistema RNS está compuesto de una serie de n números coprimos por pares llamado conjunto de módulos que cumple con $m_i \equiv 1 \; mod \; m_j \mid \forall i, j \in \{m_1 < m_2 < \cdots < m_n\}$. Los módulos coprimos

permiten establecer un Isomorfismo entre el mapeo de $X \in [0, P)$ y la representación RNS de $x_1, x_2, .. x_n$, con ello, la secuencia de operaciones que se efectúan en una estructura puede ser mapeada a la otra.

Un número entero $X \in [0, P)$, donde $P = \prod_i^n m_i$ es conocido como rango legítimo, puede ser representado en este sistema por medio de los residuos resultantes de efectuar una división entre los n módulos coprimos; talque $X \xrightarrow{RNS} x_1, x_2, x_3, \dots, x_n$. Los residuos se definen de la siguiente manera $X \equiv x_i \; mod \; m_i$ y se restringen a $x_i \in [0, m_i)$.

Para recuperar el número se utiliza el CRT, este establece que si los n módulos son coprimos a pares entonces un número X representado en RNS tiene una única solución que satisface todas las congruencias del sistema. Esto se puede ver más detalladamente en el siguiente sistema de ecuaciones modulares:

$$X \equiv x_1 \; mod \; m_1$$

$$X \equiv x_2 \; mod \; m_2$$

…

$$X \equiv x_n mod \; m_n$$

Para recuperar la información original se puede emplear la siguiente formula:

$$X = \left(\sum_{i=1}^{k} x_i \; P_i b_i \right) mod \; P \qquad (3)$$

$\forall \; 1, \dots, \mathrm{n}$ donde $P_i = P/m_i$ y b_i es el multiplicativo inverso de P_i modulo p_i.

2.3.6 Sistema Redundante de Número Residuo (RRNS)

Este sistema es una adaptación de RNS que agrega un conjunto de módulos adicionales para poder introducir redundancia al sistema, además, permite que RNS tenga características de sistema umbral (k, n).

Los cambios principales radican en establecer k módulos (rango legitimo) y adicionalmente r módulos (rango ilegitimo) de redundancia, teniendo como resultado $n = k + r$ módulos. Recordando que todos deben ser coprimos a pares para tener un conjunto $\{m_1 < m_2 < \ \ldots < m_k < m_{k+1} < \cdots < m_n\}$.

Ahora, el entero $X \in [0, P')$ donde $P' = \prod_i^k m_i$ se denomina rango dinámico conformado por los primeros k módulos. El número en RRNS queda de la siguiente manera $X \xrightarrow{RRNS} \{x_1, x_2, \ldots, x_k, x_{k+1}, \ldots, x_n\}$.

Al manejar estas consideraciones se puede recuperar X empleando CRT con un subconjunto de tamaño menor a k de cualquiera de los residuos (pedazos) de los n generados originalmente. Tal que se tiene un conjunto de k pedazos $\{x_{i_1}, x_{i_2}, \ldots x_{i_k}\} \subseteq \{x_1, x_2, \ldots, x_n\}$, así como también su correspondiente subconjunto de módulos $\{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\} \subseteq \{m_1, m_2, \ldots, m_n\}$. Si fallan $y \leq r$ de los CSPs entonces se puede recuperar la información de los restantes $k + r - y$ CSPs. La información puede ser recuperada con:

$$P' = \prod_j^k m_{i_j}$$

$$X = \left(\sum_{j=1}^{k} x_{i_j} \, {P'}_j \ \ {P'^{-1}}_j \right) mod \, P', \tag{4}$$

donde ${P'}_j \ = \frac{P'}{m_{i_j}}$; $\ {P'^{-1}}_j$es el inverso multiplicativo, talque $\left({P'}_j \ {P'^{-1}}_j \right) mod \, m_{i_j} = 1$.

Además, agregar redundancia al sistema RNS permite detectar y corregir errores hasta la multiplicidad de r y $\left\lfloor \frac{r}{2} \right\rfloor$, respectivamente. Esta ventaja hace de RRNS un sistema con cualidades atractivas para el entorno distribuido, sin necesidad sistemas adicionales.

2.3.7 Sistema Redundante de Número Residuo con Aproximación de Rango (AR-RRNS)

Este método es similar en todo aspecto al RRNS con una particularidad durante la decodificación, Chervyakov et al. (2019) introdujeron un método de conversión de RRNS a binario basado en la aproximación de rango (AR, del inglés, *Approximation of Rank*). Este método reduce el tiempo de cómputo

al momento de la decodificación a costa de la cantidad de memoria utilizada. El incremento de memoria se debe a un mayor número de variables almacenadas durante el proceso de decodificación.

Parte de este método consiste en una modificación de la fórmula anterior para recuperar X:

$$X = \sum_{i=1}^{n} P_i \left(P_i^{-1} \, mod \, m_i\right) x_i - r_x . P \ , \tag{5}$$

donde:

$r_x = \left\lfloor \sum_{i=1}^{n} \frac{P_i^{-1} mod \, m_i}{m_i} x_i \right\rfloor , P = \prod_{i=1}^{n} m_i \, , P_i = P/m_i \, , P_i^{-1}$ es el inverso multiplicativo

La propuesta de Chervyakov et al. (2019) fue reemplazar r_x por R_x. A nivel computacional el cambio implica el reemplazar operaciones de alta precisión por operaciones de aproximación de menor coste. Con lo cual la propuesta se basa en el Teorema: Si $N = \lceil \log_2 \rho \rceil$ entonces $r_x = R_x$ o $r_x = R_x - 1$ donde ρ = $\sum_{i=1}^{n} p_i - n$.

$$R_x = \left\lfloor \sum_{i=1}^{n} k_i x_i / 2^N \right\rfloor , \tag{6}$$

Se puede desarrollar un ejemplo para ver con mayor detalle el proceso general y la aportación de utilizar la aproximación de rango:

Ejemplo 1:

Sea un conjunto de módulos $m_1 = 2, m_2 = 3$ y $m_3 = 5$. El rango dinámico estará dado por $P = m_1 * m_2 * m_3 = 30$. La conversión RNS es $X = 10 \xrightarrow{RNS} \{0, 1, 0\}$ y se calculan las constantes que serán resguardas en memoria:

$P_1 = P/m_1 = 15$, $P_2 = P/m_2 = 10$ y $P_3 = P/m_3 = 6$;

$(P_1^{-1} \, mod \, m_1)P_1 = 15$, $(P_2^{-1} \, mod \, m_2)P_2 = 10$ y $(P_3^{-1} \, mod \, m_2)P_3 = 6$

$$\rho = -3 \ + 2 + 3 + 5 = 7$$

$$N = \lceil log_2 7 \rceil = 3$$

$k_1 = \left\lceil (P_1^{-1} mod\, m_1) 2^3 / m_1 \right\rceil = 4$, $k_2 = \left\lceil (P_2^{-1} mod\, m_2) 2^3 / m_2 \right\rceil = 3$ y $k_3 = \left\lceil (P_1^{-1} mod\, m_1) 2^3 / m_3 \right\rceil = 2$

Se pre calculan los siguientes valores para calcular el rango:

$$\sum_{i=1}^{3} k_i x_i = 4 * 0 + 3 * 1 + 2 * 0 = 3$$

Utilizando la ecuación de R_x tenemos:

$$R_x = \left\lfloor \sum_{i=1}^{n} k_i x_i / 2^N \right\rfloor = 0$$

Finalmente se calcula el valor de X:

$$X = \sum_{i=1}^{n} P_i \left(P_i^{-1} \, mod \, m_i \right) x_i - r_x . P_i \; = 15 * 0 + 10 * 1 + 6 * 0 - 30 = 10$$

Nota: Cuando $X < 0$, es un valor negativo, se realiza la siguiente operación $X = X_{neg} + P$.

Además, el utilizar AR-RRNS permite detectar desbordamientos y errores; e incluso corregir los errores. Esto es aplicado en Babenko et al. (2017) donde AR-RRNS es propuesto para redes P2P las cuales sufren de padecimientos similares que el cómputo en la nube al tener una naturaleza distribuida.

En caso de que el secreto a guardar sea mucho mayor que lo que permite el rango dinámico de los k módulos seleccionados, entonces la información se divide en bloques de la misma longitud de bits para representar el sistema dinámico seleccionado.

Capítulo 3 Redes apuntadoras

En este capítulo se introducen conceptos teóricos pertinentes a modelos cognitivos de redes neuronales (NN, del inglés Neural Networks). Posteriormente, se profundiza en una clase particular de NN llamada redes apuntadoras, parte fundamental del objeto de estudio del presente trabajo. Finalmente, se analizan diversos tópicos relacionados al aprendizaje por refuerzo.

3.1 Introducción

Las NN representan una clase de modelos computacionales inspirados en las neuro ciencias. El desarrollo de estas estructuras se puede describir a partir de tres principales olas o generaciones identificadas por Goodfellow et al. (2016): cybernetics (1940), redes conexionistas o neuronales (1960) y aprendizaje profundo (2006).

El perceptrón fue la primera implementación de NN, Rosenblatt (1957) lo describió en un inicio como: "*Un sistema que puede localizar memorias relevantes, sin recurrir a un proceso de búsqueda secuencial*. *Se basa más en principios probabilísticos, que en principios determinísticos. Y gana confiabilidad de las mediciones estadísticas de grandes poblaciones de elementos*". El modelo de Rosenblantt fue el primero en aprender sus propios parámetros; de una manera rudimentaria.

Posteriormente, se presentó ADALINE una versión mejorada donde se dio la introducción de una función de activación lineal y un cuantificador. ADALINE empleó el descenso de gradiente simple (GD, por sus siglas en inglés) para ajustar los pesos del modelo (Widrow y Hoff, 1960). Una limitante de estos modelos fue su incapacidad de abordar problemas no linealmente separables (Minsky y Papert, 1969).

La segunda ola en el desarrollo de las NN comenzó por la influencia del movimiento conocido como conexionistas o procesamiento distribuido paralelo. Este movimiento planteaba ideas complicadas de las ciencias cognitivas, la principal motivación era que una gran cantidad de computación simple puede alcanzar inteligencia cuando estas trabajan en conjunto de forma conectada (Goodfellow et al., 2016).

Esta segunda ola introdujo conceptos y algoritmos relevantes para el área, como:

- Representación distribuida: cada entrada a un sistema es representada por muchas características y cada característica involucra muchas entradas del sistema (Hinton, 1984).

- Propagación hacia atrás (Back-Propagation): Calcula el gradiente de una NN propagando el error hacia atrás entre las capas con respecto a los parámetros del modelo, lo que permite el entrenamiento de redes multi capa (Rumelhart et al., 1986).

- Long Short-Term Memory (LSTM): Es una estructura de NN que permite procesar largas secuencias y manejar dependencias temporales (Hochreiter y Schmidhuber, 1997).

El término de la segunda ola se debió a no poder cumplir con las promesas que se plantearon a los inversores de capital, justo cuando se daba el desplome de las ".com" y que otros enfoques proporcionaban mejores resultados. Sin embargo, las NNs continuaron bajo el desarrollo del laboratorio CIFAR (Goodfellow et al., 2016).

La demostración de Hinton et al., (2006) sobre como las redes de gran dimensión pueden ser entrenadas utilizando cierta inicialización en los pesos dio inicio a la tercera ola, además del desarrollo de hardware dedicado posteriormente. Un incremento en la investigación resulto en nuevas arquitecturas y estructuras destacadas en las NNs como lo son:

- **Redes residuales:** permiten entrenar redes muy profundas utilizadas para clasificación y minimizan el efecto de desvanecimiento de gradiente (He et al., 2015a).

- **Unet:** enfocadas a la segmentación, escalamiento y eliminación de ruido en imágenes, entre muchas otras aplicaciones (Ronneberger et al., 2015).

- **Autocodificador variacional:** generación de datos complejos (Kingma y Welling, 2014).

- **Redes generativas antagónicas:** generación de datos y modelos (Goodfellow et al., 2014).

- **Transformers:** Sistemas de segmentación, secuencia a secuencia, entre otros (Vaswani et al., 2017).

Una NN busca aproximar una función $f^{\#}$ tal que si se tienen una entrada x, parámetros internos θ y una representación y deseada (conocida también como objetivo) entonces se pueda mapear $y = f^{\#}(x; \theta)$. Si la NN no contiene ciclos es llamada NN alimentada hacia delante

La entrada x de la NN sufre de transformaciones no lineales ϕ tal que $\phi(x)$. Incluso, ϕ puede ser otro modelo con parámetros θ_ϕ de manera que $\phi(x; \theta_\phi)$. Las NN pueden verse como una composición de múltiples funciones donde la cantidad de composiciones es la profundidad de la red. La primera capa o función es la entrada del modelo, la última capa representa la salida y las capas intermedias se denominan ocultas. Una red puede ser poco profunda (del inglés, shallow) o profunda (del inglés, deep), la diferencia radica en el número de capa oculta del modelo.

De acuerdo a Bishop (2006), el aprendizaje de máquina (Machine learning) puede ser categorizado en:

- **Aprendizaje supervisado**: Se tiene una tupla (entrada, objetivo) y el entrenamiento consiste en hacer que el predictor (modelo) "aprenda" a aproximar el objetivo a partir de la entrada.

- **Aprendizaje no supervisado**: El modelo extrae información relevante de los datos durante el entrenamiento y posteriormente se utiliza para inferir.

- **Aprendizaje por reforzamiento**: Se aprende mediante la exploración y explotación de las acciones de los estados hasta encontrar las acciones óptimas que generen el mayor beneficio.

Las NN utilizan el aprendizaje supervisado y el aprendizaje por reforzamiento. Es necesario estudiar la estructura general de las NNs para entender el funcionamiento del aprendizaje.

3.2 Neuronas o Unidades de procesamiento

Las neuronas o unidades de procesamiento son transformaciones afines con parámetros que se definen mediante el entrenamiento de la NN, seguido de una función no lineal (véase Figura 1). La agrupación de varias neuronas conforma una capa, las capas son representadas por medio de una estructura de vectores:

$$\boldsymbol{h} = \sigma(\boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b}) \tag{7}$$

Donde:

- $\boldsymbol{h}$ es un vector $\mathbb{R}^p$,
- σ es una función no lineal,
- $\boldsymbol{x}$ es un vector $\mathbb{R}^q$ que define la entrada,
- $\boldsymbol{W}$ es un matriz $\mathbb{R}^{p\times q}$que establece las transformaciones a fines,
- $\boldsymbol{b}$ es un vector $\mathbb{R}^p$ que evita el sesgo (en inglés, *bias*).

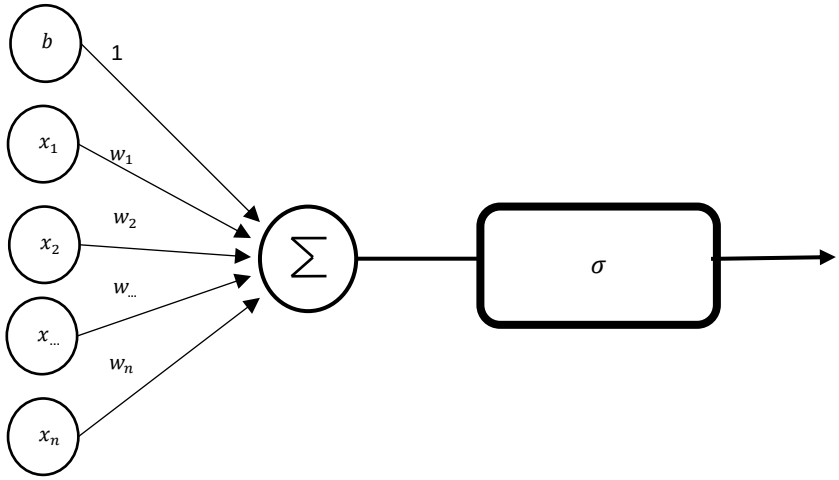


Figura 1. Representación de una unidad o neurona\perceptrón con sesgo (denotada con una arista de peso 1) y una función de activación no lineal σ.

3.3 Funciones no lineales utilizadas

La función no lineal más utilizada es la rectificadora y se motiva su uso en capas ocultas. La unidad que la utilice es nombrada como unidad lineal rectificada (*ReLU*, por sus siglas en inglés), definida como $z = max(0, x)$.

Algunas de las funciones no lineales más utilizadas son:

- Unidad lineal rectificada *ReLU,*

  - Leaky ReLU,
  - Parametric ReLu o PReLU,
  - MaxOuts,
- Sigmoide,
- Logística,
- Tangente hiperbólico,
- SoftPlus,
- SoftMax.

El diseño para las unidades ocultas cuenta con poco fundamento teórico como guía y son elegidas de acuerdo con su desempeño (Goodfellow et al., 2016).

Funciones como la sigmoide, logística, tangente hiperbólica, entre otras, alcanzan puntos de saturación lo que dificulta el entrenamiento por GD; por tanto, no se recomienda su uso en las unidades ocultas. Existen modelos donde se justifica su uso como en el caso de las LSTM (Hochreiter y Schmidhuber, 1997). Estas funciones son recomendadas como parte de la capa de salida de la NN, algunas permiten representar probabilidades y distribuciones. Por ejemplo, la función SoftMax permite calcular una distribución normalizada a partir de las unidades de salida,

$$softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{8}$$

3.4 Redes convolucionales

LeCun (1989) presento las redes convolucionales y son parte fundamental de distintivas implementaciones de NN modernas. En la práctica, la convolución esta implementada como una correlación cruzada para

ahorrar memoria; esto permite aplicar una gran cantidad de convoluciones con tensores de menor tamaño (Kernel) y reutilizarlos al desplazarlos a través de la entrada. La formulación discreta sobre un tensor 2D puede representarse con la entrada $I \in \mathbb{R}^2$, el kernel $K \in \mathbb{R}^2$, el operador de convolución $*$ y los índices de la entrada donde se realizará la convolución (i, j), (véase Figura 2).

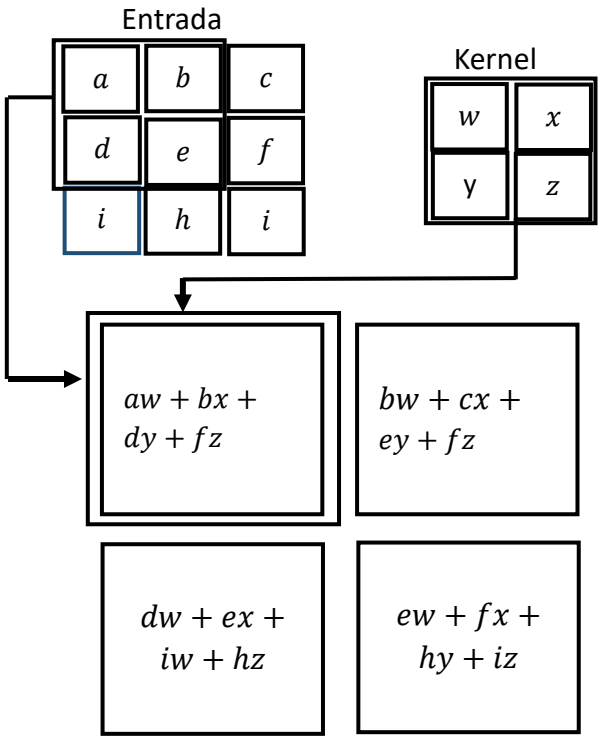


Figura 2. Representación de una operación convolucional 2D con restricción a sus valores válidos.

$$S(i,j) = (I * K)(i,j) = \sum_{m,n} I(\mathrm{i}+m, \mathrm{j}+n) K(m,n) \tag{9}$$

Las ventajas que ofrecen esta clase de redes y sus generalizaciones son:

- Interacciones dispersas (en inglés, sparse interactions).
- Compartición de parámetros.
- Representación equivariante (Robusto a traslaciones).

- Procesamiento de datos sin una medida fija, algo que una multiplicación convencional de matrices no podría.

- Trabaja con datos claramente estructurados en topología de rejilla.

- Menor cantidad de pesos/parámetros, en función del tamaño del Kernel.

Las redes convolucionales son utilizadas de diversas maneras; pueden ser configuradas para aumentar o disminuir la dimensionalidad de datos, esto permite generar unas nuevas representaciones de los datos.

3.5 Descenso de gradiente (GD) y función de costo

Las NN modernas son entrenadas con descenso de gradiente (GD). En forma simple, un GD se caracteriza por minimizar una función de costo J que evalúa un modelo h parametrizado por $\theta \in R^n$, GD puede encontrar mínimo (o máximos) local en funciones no convexas y mínimos globales para funciones convexas, todo esto por medio de ajustar los parámetros en sentido contrario del gradiente de la función de costo $\nabla_\theta J\big(h(\mathrm{x};\theta)\big)$, véase Figura 3 (Ruder, 2017).

Para controlar y evitar que el GD oscile sin control y en consecuencia garantizar la convergencia, se utiliza un hiper parámetro η el cual controla la "tasa de aprendizaje" (en inglés, *learning rate)*. Sí η es demasiado grande entonces el aprendizaje será muy agresivo y probablemente pasemos por alto algún mínimo, de otra forma, sí η es demasiado pequeño el tiempo de aprendizaje puede ser excesivo. Existen distintas implementaciones del GD que buscan modificar η activamente durante el entrenamiento. La versión básica del GD se define como:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(h(\mathrm{x};\theta_{t-1})) \qquad (10)$$

Donde:

- θ es el vector que representa los pesos del modelo,

- t es la época o iteración,

- η es el hiper parámetro que determina el tamaño del paso en cada iteración,

- $\nabla_\theta J(\ \ )$ es la derivada de primer orden con respecto a los pesos/parámetros del modelo.

La función de costo J genera un real y puede ser diseñada para tener su valor mínimo en alguna función de nuestro interés, que la red pueda representar. Seleccionar un conjunto de pesos para la red neuronal es análogo a escoger una función parametrizada (Goodfellow et al., 2016).

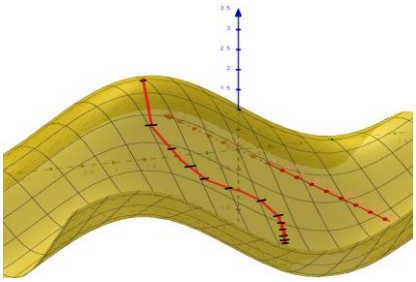


Figura 3. Representación del GD (línea roja) al minimizar la función de costo y dirigirse a un mínimo local, el comportamiento del GD depende del algoritmo y sus parámetros.

Debido a su importancia es necesario utilizar funciones de costo que trabajen en conjunto con las unidades de salida. Algunas funciones no lineales utilizadas con la salida tienden a saturarse o no estar definidas en ciertos intervalos, tal es el caso de la función logística, que se satura en valores extremadamente grandes o negativos (véase Figura 4).

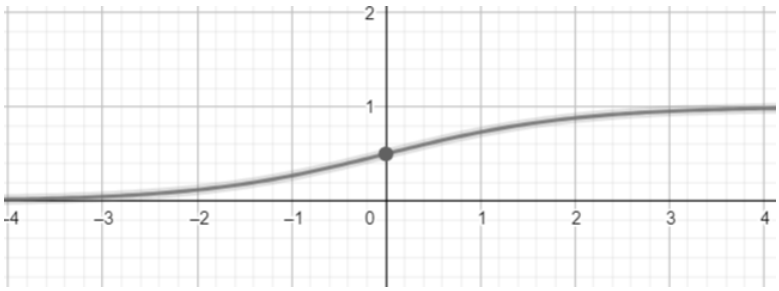


Figura 4. Gráfica de la función logística. Grandes magnitudes pueden saturar la variable en 1 o 0 causando un aprendizaje deficiente o nulo de la red.

Al saturarse una función la técnica de descenso de gradiente no puede ajustar los pesos/parámetros del modelo apropiadamente; resultando en un entrenamiento deficiente.

Por ejemplo, en aprendizaje supervisado, cuando la salida de una NN está dada por una sigmoide se puede utilizar la verisimilitud logarítmica negativa, el logaritmo de esta función tiene la propiedad de anular el efecto de la función exponencial que es parte de la sigmoide e impide que se sature, resultando en un mejor entrenamiento y convergencia hacia un mínimo local.

3.6 Versiones de SGD

Existe una serie de implementaciones distintas del descenso de gradiente, las versiones más utilizadas vienen incluidas en los frameworks utilizados por la comunidad de investigadores/desarrolladores.

Hay tres variaciones principales del DG, la diferencia radica en la cantidad de datos utilizados en el cálculo del gradiente (Ruder, 2016):

- **Batch or Vanilla Gradient Descent (BGD):** Toma todos los datos del lote para para procesar el gradiente y ajustar los parámetros del modelo.

- **Stochastic Gradient Descent (SGD):** Calcula el gradiente para cada elemento del lote, e ínvidamente actualiza los parámetros del modelo. Sin embargo, tiene una alta varianza en durante el aprendizaje y no converge muy bien.

- **Mini Batch Descent Descent (MBGD):** Toma pequeños lotes entre rangos de 56 a 256, típicamente, para procesar el gradiente y ajustar los parámetros del modelo. En la actualidad, este algoritmo se utiliza en la mayoría de los trabajos.

GD tiene algunas limitaciones ampliamente conocidas, se han identificado distintos problemas como:

- Elegir un paso de aprendizaje η adecuado.

- En ocasiones cada parámetro del modelo necesita un paso de aprendizaje η distinto.

- Dificultad de escapar de puntos de silla (Puntos o zonas donde el gradiente es cero o cercano a él), y además no es un mínimo local o global.

- Escapar de mínimos locales poco prometedores.

Debido a estos inconvenientes y otros más, se siguen desarrollando estrategias que intentan superar los retos del GD. En general, no existe una versión de GD que garantice convergencia a un mínimo global a menos que J sea convexa, lo que no suele pasar. Algunas versiones modifican el comportamiento del paso del aprendizaje y\o el gradiente de forma activa durante el entrenamiento para mejorar el desempeño, otros consideran el comportamiento histórico del gradiente. Las implementaciones más conocidas son:

- Momentum Gradient Descend - MGD (Qian, 1999).
- Nesterov Accelerated Gradient - NAG (Nesterov, 1983).
- Algoritmo de Gradiente Adaptativo - AdaGrad (Duchi et al., 2011).
- Adaptive Learning Rate Method - Adadelta (Zeiler, 2012).
- Root Mean Square Propagation - RMSProp (no publicado).
- Adaptive Moment Estimation - Adam (Kingma y Ba, 2014).
- AdaMax (Kingma y Ba, 2014).
- Nadam (Dozat, 2016).

Se realiza una breve descripción del funcionamiento de Adam debido a su amplio uso en el área.

EL nombre Adam proviene de estimación del momento adaptativa. Adam utiliza un mecanismo parecido a RMSProp y Adagrad donde el aprendizaje es adaptable, esto quiere decir que toma en cuenta información del comportamiento del gradiente durante varias iteraciones para adaptarlo conforme avanza el aprendizaje. Además, el paso de aprendizaje adaptable es individual con respecto a cada parámetro del modelo, con lo que se reducen los parámetros con alta varianza en el gradiente y se confía en parámetros más estables. Esta ventaja a cambio de sacrificar un poco de memoria para almacenar los datos necesarios para calcular el paso de aprendizaje de manera individual (Kingma y Ba, 2014).

A pesar del buen funcionamiento de Adam en muchas áreas, no es el optimizador absoluto porque en algunos problemas no llega a generalizar tan bien como MGD.

3.7 Inicialización de pesos

El ajuste de pesos/parámetros del GD es fundamental durante el entrenamiento. Definir los valores iniciales de los pesos es difícil debido a que los algoritmos son fuertemente afectados por esta decisión, Goodfellow et al. (2016) identifica varios puntos relevantes respecto la inicialización de los pesos:

- Los valores de inicio pueden determinar si el entrenamiento converge.

- Un nuevo conjunto de pesos puede resultar en un desempeño distinto, aun con el mismo modelo y datos de entrenamiento.

- Los pesos no se pueden inicializar con el mismo valor, esto es conocido como quiebre de simetría y es una de las pocas reglas que se conocen con certeza en la inicialización de los pesos. Neuronas que reciben la misma entrada, función de activación y pesos se actualizan con el mismo valor durante el entrenamiento, un comportamiento no deseado.

Existen múltiples heurísticas para generar valores aleatorios que rompen con la simetría de los pesos. La versión de pytorch 1.7 cuenta con inicializadores de números y constantes:

- **Constante**: Todos los pesos iniciales son una constante.

- **Unos**: Todos los pesos iniciales son unos.

- **Ceros**: Todos los pesos iniciales son ceros.

- **Distribución uniforme**: Los pesos son tomados de una distribución uniforme $\mathcal{U}(a, b)$.

- **Xavier (Glorot) uniforme**: Los pesos son tomados de una distribución uniforme $\mathcal{U}(-a, a)$, $a = factor + \sqrt{6 \,/\, (conexiones_{entrada} + conexiones_{salida})}$ (Glorot y Yoshua, 2010).

- **Xavier (Glorot) normal**: Los pesos son tomados de una distribución normal $\mathcal{N}(0, std^2)$, $std = factor + \sqrt{2 \,/\, (conexiones_{entrada} + conexiones_{salida})}$ (Glorot y Yoshua, 2010).

- **Kaiming uniforme**: Los pesos son tomados de una distribución uniforme $\mathcal{U}(-a, a)$ con $a = factor + \sqrt{3 \,/\, modo}$ donde $modo$ define las conexiones de entrada o conexiones de salida, las conexiones de entrada preservan la magnitud de la varianza de la propagación hacia delante y las conexiones de salida preservan las magnitudes del paso hacia atrás del cálculo de los gradientes (He et al., 2015b).

- **Kaiming normal**: Los pesos son tomados de una distribución normal $\mathcal{N}(0, std^2)$ con $std = factor \sqrt{factor \,/\, modo}$ donde $modo$ se configura y tiene el mismo efecto Kaiming uniforme (He et al., 2015b).

- **Disperso**: Una matriz dispersa es definida con una distribución normal $\mathcal{N}(0, 0.01)$ para los pesos diferentes a cero (Saxe et al., 2014).

- **Ortogonal**: Usa los valores de una matriz (semi) ortogonal.

El uso de inicializadores para unos, ceros y constantes debe estar justificado.

3.8 Redes recurrentes

Las redes recurrentes permiten manejar secuencias con dependencias distantes entre los elementos, algo que las redes tradicionales no pueden hacer. Para manejar estas dependencias, los parámetros en distintos puntos de la NN se comparten durante la propagación hacia delante.

En una red recurrente, la salida de una etapa es resultado de aplicar la función de la etapa a las salidas previas producidas por la misma etapa, creando de cierta manera un bucle. Una red profunda con parámetros compartidos en distintas partes del modelo es el resultado de representar el grafo con ciclos de la red con un grafo sin cíclicos (acíclico), véase Figura 5.

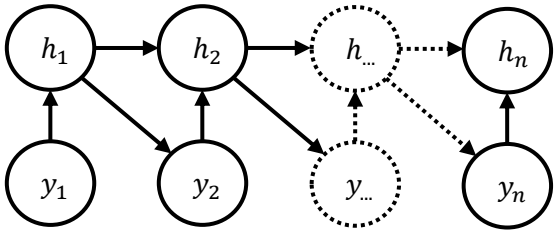


Figura 5. Representación en forma de grafo acíclico de la red recurrente donde los estados generan y_i a partir de (h_{i-1}, y_{i-1}). Los resultados previos tienen influencia en los resultados actuales debido a los estados internos.

"Los ciclos representan la influencia del valor presente de una variable en su propio valor a futuro" (Goodfellow et al., 2016).

En general, una red recurrente puede ser representada con la formula siguiente:

$$h_t = f(h_{i-1}, x_t; \theta) \tag{11}$$

Donde:

- h_{i-1}representa el vector salida con el estado de las unidades escondidas en el tiempo t.
- x_t define un vector en el tiempo t que condiciona el estado de salida.
- θ parámetros que parametrizan la recurrencia, nótese que este no depende del tiempo t.

Cada vector h se puede introducir a una red\función adicional para mapearlo a una representación buscada. El compartir parámetros se sustenta en la relación de la probabilidad condicional t dado $t + 1$, no solo se depende de t (propiedad estacionaria). Sin embargo, en ocasiones se busca una única dependencia de t, en estos casos, t se suministra como una entrada más al modelo.

Existen diversas maneras de manipular la recurrencia para inducir diferentes probabilidades condicionales con respecto a la entrada. Por ejemplo, suministrar elementos adicionales a la entrada de la recurrencia agrega un sesgo a la función.

3.8.1 Redes Long-Short Term Memory (LSTM)

La red LSTM fue diseñada para evitar el problema de desvanecimiento del gradiente cuando se manejan secuencias grandes. Esta estructura recurrente emplea puertas (*gates*, en inglés) que permite olvidar o almacenar información relevante de la secuencia de entrada (Bello et al., 2016).

La red LSTM de Hochreiter y Schmidhuber (1997) está conformada de la siguiente manera:

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big) \tag{12a}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{12b}$$

$$\tilde{C} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{12c}$$

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{12d}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{12e}$$

$$h_t = o_t * tanh(C_t) \tag{12f}$$

En una red LSTM, la entrada está dada por tres vectores x_t, h_{t-1} y C_{t-1} que representan la entrada actual, la salida y el estado interno anterior de la LSTM, respectivamente. La compuerta del olvido utiliza una función sigmoide σ para decidir el grado de olvido en un rango de [0, 1] donde 0 es olvidarlo todo y 1 es recordar todo, vea ecuación (12a). La compuerta de entrada es similar a la compuerta del olvido pero el vector define el grado de entrada de nuevos valores introducidos en el tiempo t, vea ecuación (12b). En la ecuación (12c), los valores candidatos son valores de entrada con un preprocesamiento parametrizado aplicado. La ecuación (12d) genera el nuevo estado interno C_t de la LSTM al combinar el estado anterior C_{t-1}, el vector de valores candidatos $\tilde{C}$ y las respectivas compuertas. La salida de la ecuación (12e) proporciona un resultado similar a (12a) y (12b). Finalmente, en la ecuación (12f) se aplica una función tangente hiperbólica al estado interno C_t y se multiplica por el vector correspondiente a su compuerta (12e).

Las redes LSTM bidireccionales constan de dos redes LSTM, una red recibe la secuencia $\boldsymbol{x}$ en forma original mientras que la segunda red recibe la secuencia $\boldsymbol{x}$ ordenada inversamente, posteriormente sus respectivas salidas son concatenadas o agregadas a una sola salida.

3.8.2 Generalizaciones de redes recurrentes

Sutskever et al. (2014) definieron la estructura de secuencia a secuencia también conocido como codificador-decodificador. Esta estructura de NN permite inferir una secuencia dada otra donde la longitud de la segunda está en función de la primera.

El codificador recibe la secuencia de entrada $x = (x_1, x_2, \dots, x_n)$ y la codifica a un vector de contexto C de dimensión fija, una versión compacta de x y no hay manera de saber la dimensión ideal de C. Posteriormente, el decodificador es condicionado con el vector C y empieza a inferir la segunda secuencia, véase Figura 6.

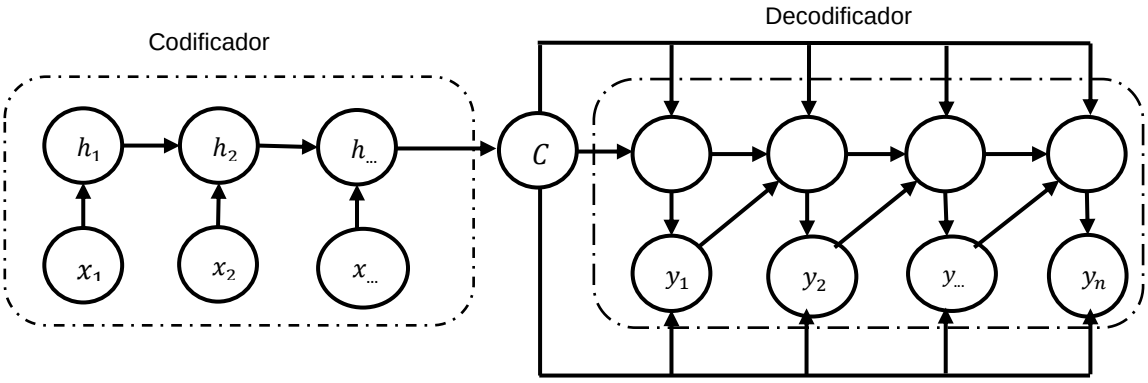


Figura 6. Representación de la arquitectura codificador-decodificador donde los elementos x_i son codificados en un vector C que es utilizado múltiples veces para genera la secuencia de salida.

Usar una representación C de dimensión fija para una entrada x de n elementos podría no representarla para una correcta inferencia. Implementar una red lo suficientemente profunda podría resolver este problema (Cho et al., 2014) pero se tendrían tiempos más largos de entrenamiento e inferencia. En general un entrenamiento más complicado.

El mecanismo de atención permitió superar esta limitación sin aumentar la red en gran medida (Bahdanau et al., 2015), la implementación del mecanismo de atención enriquece la consulta con información de la secuencia de entrada $\boldsymbol{x}$, el cual consiste en los siguientes pasos:

1. Cada elemento $\boldsymbol{x}$ es codificado y almacenado para su consulta,

2. El mecanismo de atención parametrizado recibe la consulta q_i y los elementos codificados de $\boldsymbol{x}$. El mecanismo sustituye q_i por una consulta q_i' conformada por la contribución ponderada de los elementos codificados de $\boldsymbol{x}$ y el elemento de consulta original q_i, véase (13).

$$u_j^i = v^T tanh\big(W_1\, e_j + W_2\, q_i\big), j \in (1,2,\dots,n) \tag{13.a}$$

$$a_j^i = softmax\big(u_j^i\big), j \in (1,2,\dots,n) \tag{13.b}$$

$$q_i' = \sum_{j=1}^{n} a_j^i\, e_j \tag{13.c}$$

Donde

- $W_1, W_1 \in \mathbb{R}^{d*d}$ son tensores cuadrados (matrices cuadradas) de atención,
- $v^T \in \mathbb{R}^d$ es un tensor (vector columna) de atención,
- $d \in \mathbb{N}^+$ es la dimensión de los estados ocultos,
- $u^i = \{u_1^1, u_2^1 \dots, u_n^1\}$, es el vector de salida del mecanismo de atención no normalizado,
- $softmax$ es la función que normaliza la distribución de los elementos $\boldsymbol{u}^i$, vea ecuación (8),
- q_i es el vector de consulta, usualmente la salida del decodificador en la estructura secuencia a secuencia (normalmente la salida de una LSTM $(12.f)$,
- a_j^i es el nivel de atención al elemento codificado e_j en la inferencia del elemento i de la segunda secuencia,
- q_i' es el nuevo vector consulta que contiene información de todos los elementos de la entrada.
- e_j es elemento x_i codificado.

3.8.3 Modos de indicar el fin de una secuencia

En muchos casos no hay manera a priori de saber la longitud de la cadena que se busca inferir, por ello es necesario tener un método para decidir el fin de una cadena, alguna de las opciones recopiladas por Goodfellow et al. (2016) para lograr esto son:

- **Símbolo especial:** Un símbolo que al ser inferido/seleccionado indica el fin de la secuencia.

- **Señal de alto:** Implementar una salida adicional en la red para indicar la probabilidad de detener la inferencia de la cadena.

- **Predicción:** Predecir la longitud de la cadena a inferir a priori con base en la entrada. Se introducen la cantidad de elementos inferidos y la cantidad máxima en la entrada para sesgar la inferencia de la cadena.

3.9 Redes apuntadoras (Ptr-Net)

La red apuntadora (Ptr-Net) (Vinyals et al., 2015) surgió de los modelos codificador-decodificador (Sutskever et al., 2014) y del sistema de atención (Bahdanau et al., 2015). El modelo codificador-decodificador mapea dos secuencias de diferente longitud. La longitud de la primera cadena no condiciona la longitud de la segunda pero si está en función de ella. Los sistemas de atención aportan información contextual de los elementos de la entrada para enriquecer la consulta.

La mayor contribución de la red Ptr-Net es poder seleccionar elementos de la entrada por medio de un diccionario de identificadores (*tokens)* del mismo tamaño; esto implica que no es necesario definir a priori las clases para la inferencia. El análisis de la Ptr-Net para la traducción de texto en diferentes idiomas ilustra el funcionamiento de estas redes.

Los modelos clásicos de traducción de texto utilizan entrenamiento supervisado, los valores de entrenamiento de la red son tuplas valor-objetivo $\left(l^{inp}, l^{tar}\right)$ donde l^{inp} define la entrada como una cadena de texto del idioma original y l^{tar} es la cadena de salida en el idioma objetivo que aproxima la entrada. Por lo tanto, las clases para la inferencia son estáticas y están conformadas por el diccionario del idioma objetivo $dict_{objetivo}$ tal que $l_i^{tar} \in dict_{objetivo}$. Los modelos clásicos no funcionan para resolver

problemas combinatorios porque no es posible representar cada una de las posibles entradas en una clase en la salida, donde las clases son los elementos de la entrada.

Para solucionar el problema considerado anteriormente, la red Ptr-Net modifica la estructura codificador-decodificador para introducir un mecanismo de atención similar a (13). Este mecanismo genera una distribución sobre los elementos de entrada y permite seleccionarlos como si de un diccionario único para cada instancia del problema se tratara. La probabilidad de elegir un elemento de la entrada está dado por:

$$u_j^i = v^T tanh\big(W_{ref}\, ref_j + W_q\, q_i\big), j \in (1, \dots, n) \tag{14.a}$$

$$p\big(l_i^{out} \big| l_1^{out}, l_2^{out}, \dots, l_{i-1}^{out}, l^{(inp)}\big) = softmax\big(u^i\big) \tag{14.b}$$

Donde:

- $W_{ref}, W_q \in \mathbb{R}^{d*d}$ son tensores cuadrados (matrices cuadradas) de atención,
- $v^T \in \mathbb{R}^d$ es un tensor (vector columna) de atención,
- $d \in \mathbb{N}^+$ es la dimensión de los estados ocultos,
- $u^i = \{u_1^1, u_2^1 \dots, u_n^1\}$ es el vector de salida no normalizado,
- $softmax$ es la función (8) utilizada para normalizar y obtener la distribución de los elementos de u^i (tokens),
- $p(\quad)$, es la probabilidad de elegir un elemento sobre una distribución dada,
- $l^{(inp)}$ secuencia de entrada,
- l_i^{out} elemento i de salida,
- ref_j es el elemento j de la entrada codificado, como un vector, la codificación puede estar dada por $(12.f)$,

- q_i es el vector de consulta, usualmente la salida del decodificador en la estructura secuencia a secuencia (normalmente la salida de una LSTM $(12.f)$.

3.10 Framework para redes neuronales: Pytorch

PyTorch es un framework de código libre activamente soportado por Facebook AI Research y NVIDIA. Las versiones de PyTorch están alojadas en GitHub por lo que cualquier persona puede contribuir en su desarrollo. Junto con TensorFlow, ambas herramientas son los principales frameworks para el desarrollo e implementación de redes neuronales. TensorFlow es activamente soportado por Google y está construido bajo 4 principios descritos por Paszke et al. (2019):

- Se puede seguir fácilmente las directrices establecidas del lenguaje.

- Poner primero a los investigadores, los modelos deben de ser fáciles de crear y la complejidad debe ser manejada por las librerías internas, PyTorch y las APIs deben tener interfaces sencillas de utilizar.

- Proveer desempeño pragmático, el framework no sacrificar la usabilidad por el rendimiento, se intercambia hasta el 10% del rendimiento para facilitar el desarrollo de modelos. Sin embargo, el usuario siempre puede implementar un modelo complejo y obtener el 10% extra.

- Peor es mejor, es mejor tener una solución incompleta pero fácil de mantener a tener una solución compleja y difícil de mantener.

El núcleo de PyTorch este escrito con C++ para lograr un alto rendimiento, entre estas directivas están las estructuras de los datos (tensores), primitivas básicas de paralelismo y operaciones en GPU y CPU.

Una herramienta del framework denominada *eager execution* define el grafo de operaciones de la red en cada iteración. Otros frameworks establecen el grafo de operaciones antes de la ejecución y por lo tanto el grafo es estático durante todo el entrenamiento del modelo. Por el contrario, *eager execution* es un grafo dinámico que puede cambiar durante la ejecución incluso con condiciones de tipo *if*, *else*, *while*, etc. Esta característica de TensorFlow lo hace uno de los frameworks más utilizado, la actualización *TensorFlow* 2.0 buscó adaptar su funcionamiento para implementar nativamente *eager execution*. Por último, las

librerías de PyTorch y Python utilizan diferenciación automática que consiste en implementar las derivadas parciales de todas las operaciones definidas. Incluso, el usuario puede utilizar operaciones personalizadas y para ello, el framework define métodos sencillos para agregar estas operaciones.

PyTorch permite reemplazar cualquier parte que no se ajuste a las necesidades de algún proyecto, esta característica se utiliza para no imponer restricciones a los usuarios del framework.

Además, PyTorch hace uso de librerías de código abierto que han traído sorprendentes beneficios, esta situación permite los usuarios crear y contribuir en *bidings* de PyTorch y participar en el desarrollo de nuevos proyectos. Así como una comunidad muy activa donde se solucionan dudas en foros y los desarrolladores principales del proyecto contribuyen en las respuestas de manera sistemática.

Todas estas características permitieron a PyTorch posicionarse como una de las principales herramientas dentro de la comunidad científica, como lo declararon Paszke et al. (2019) “Casi abarcando un 50% de las publicaciones donde se mencionan frameworks de aprendizaje profundo de uso común en arXiv”, esto se ve reflejado en la Figura 7.

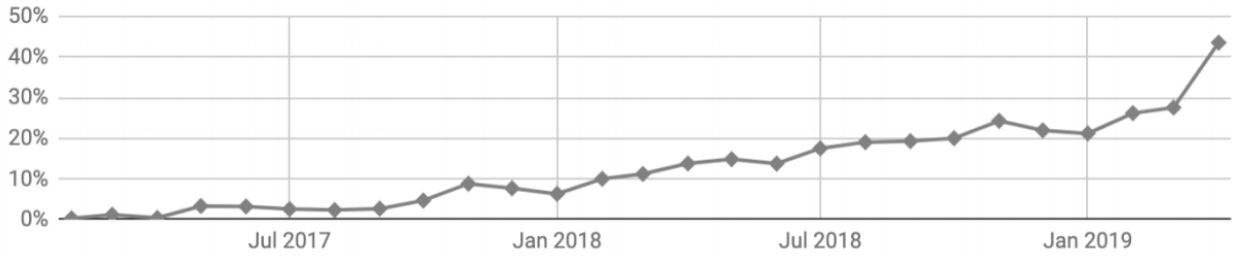


Figura 7. Porcentaje de investigaciones científicas en arXiv donde se menciona PyTorch y frameworks de aprendizaje profundo (Paszke et al., 2019).

3.11 Aprendizaje por refuerzo

El aprendizaje por refuerzo (RL, para simplificar) ha tenido un impulso en el desarrollo de muchas aplicaciones en los últimos años. La idea del RL consiste en entrenar agentes inteligentes capaces de aprender e interactuar con el ambiente mediante la toma de decisiones (acciones). Las recompensas obtenida derivado del comportamiento del agente guían el proceso de aprendizaje, es decir, la recompensa modifica el comportamiento del agente (Arulkumaran et al., 2017; Sutton y Barto, 2018).

Los modelos de aprendizaje por refuerzo presentados están basados en procesos de decisión de Markov (MDP por sus siglas en inglés) finitos. Un MDP define la clásica formalización de la selección secuencial de acciones (ver

Figura 8). Las acciones afectan las recompensas inmediatas y futuras. El uso de la recompensa para formalizar un objetivo es llamado beneficio (en inglés, *return*), este es uno de los grandes distintivos del aprendizaje por refuerzo (Sutton y Barto, 2018).

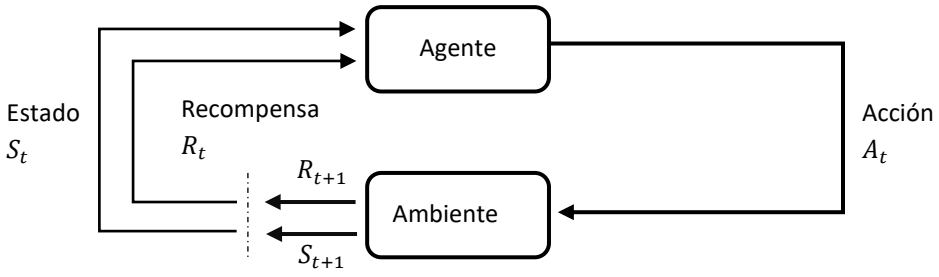


Figura 8. Interacción del agente-ambiente *en un MDP.*

3.11.1 Procesos de decisión de Markov (MDP)

La mayoría de los algoritmos de RL cumplen la propiedad de Markov, esta establece que un estado actual debe de representar todos los estados y acciones pasadas que afecten el beneficio futuro pero ser independiente de ellos. Sin embargo, esta propiedad en ocasiones es irrealista aunque es posible adaptar los problemas que no cumple la propiedad directamente.

En resumen, un MDP cuenta con las siguientes características:

- $\mathcal{S}^+$ conjunto de todos estados terminales y no terminales.
- $\mathcal{A}$ conjunto de todas las acciones.

- Dinámicas de transición, es decir, probabilidades llegar al estado S_t dado la acción A_{t-1} y estado S_{t-1}.

- Una función de recompensa.

- Un *factor de descuento* $\gamma \in [0,1]$ para una secuencia de recompensas.

En la formulación más básica, el beneficio obtenido consiste en múltiples pasos que pueden denotarse como la suma de las recompensas obtenidas donde un factor de descuento γ afecta la recompensa que no es inmediata:

$$G_t \coloneqq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \tag{15}$$

Dentro del RL existen dos enfoques esenciales, el primer enfoque enfatiza el desarrollo en base a las funciones de valor, mientras que el segundo enfoque acentúa la parametrización de la política, es decir, al agente. Las reglas y formulaciones empleadas tienen influencia de las teorías de control óptimo, en consecuencia, en las ecuaciones de Bellman.

3.11.1.1 Funciones de valor

Una estrategia RL considera que un agente puede tomar una acción valida A_t en el estado S_t para llegar al estado S_{t+1} con el valor estimado de beneficio más alto posible (recompensas), dicho estado con lleva a un valor intrínseco en futuras recompensas que podrían no ser inmediatas. Incluso, un agente podría preferir obtener una recompensa inmediata negativa en un estado S_t pero con la esperanza de una recompensa mayor a futuro. La función estado-valor V^{π} se puede establecer como el valor esperado al seguir las acciones establecidas por la política π desde un estado inicial S_t hasta un estado terminal S_T:

$$V_{\pi}(\mathrm{S}) \coloneqq \mathbb{E}_{\pi}[G_t|S_t] \tag{16}$$

Si se cuenta con el estado-valor óptimo V^* entonces se puede definir la política optima π^* para tomar la acción A_t en el estado S_t que dirija al siguiente estado de valor máximo. Sin embargo, es usual que se desconozca la distribución en la transición de los estados. Por tal motivo, es más practico involucrar las acciones disponibles en ese estado para derivar la función acción-valor, en este caso en la función además

de suministrar el estado S_t se tiene que proporcionar la acción A_t también, para posteriormente seguir la política en los demás estados:

$$Q_\pi(S_t, A_t) = \mathbb{E}_\pi[G_t|S_t, A_t] \tag{17}$$

La relación directa entre ambas funciones $V_\pi(S_t) = \max_{A(S_t)} Q_\pi(A, S_t)$ se puede interpretar de la siguiente manera: Conseguir la función estado-valor implica elegir la acción A que entrega el mayor valor esperado a partir del estado S_t. En conclusión, la función estado-valor solo se puede derivar cuando se conoce.

La ecuación de Bellman se utiliza en programación dinámica para aprender el valor de las funciones. Esto da pie al surgimiento de distintos algoritmos como SARSA (del inglés *State–Action–Reward–State–Action*) y Q-learning que utilizan técnicas como TD (del inglés, *Temporal Diference).*

3.11.1.2 Método del gradiente

Para entrenar una red neuronal se necesita definir una función J que evalúa el desempeño de la red. Esta función debe ser diferenciable con respecto a los parámetros θ de la política π, esta es la formulación clásica del descenso de gradiente, vea ecuación (10).

Los métodos de entrenamiento por refuerzo usualmente establecen funciones estado-valor o acción-valor, entonces una política π tiene la función de elegir la acción a realizar en un estado S_t para maximizar el beneficio esperado en base a estas funciones.

No obstante, la política de gradientes no usa este acercamiento para determinar la acción, el gradiente utiliza una función parametrizada que representa la política π para determinar sus decisiones, en otras palabras, solo considera el estado S_t como parámetro de entrada y no se consulta ninguna función (acción-valor o estado-valor). Sin embargo, la función estado-valor si es utilizada e incluso es aprendida para realizar una aproximación por una función parametrizada adicional; con el motivo de ser utilizada solo durante el entrenamiento para reducir la varianza. El elemento utilizado no es parametrizado.

Hay distintas implementaciones de política parametrizadas que se utilizan para aproximar la función estado-valor, por ejemplo:

- Base line: Disminuye la varianza durante el entrenamiento y mejora la convergencia.

- Actor-critico: Permite ajustar la parametrización de la función estimadora estado-valor entre un episodio y otro.

La política parametrizada en este caso está definida como la probabilidad de tomar la acción A_t dado estado S_t y los parámetros θ_π. Denotada por:

$$\pi(A_t|S_t, \theta_\pi) \tag{18}$$

Donde:

- π es la política,

- S_t es el estado,

- θ_π son los parámetros de la red neuronal que parametriza la política.

Si las acciones son discretas y finitas, las preferencias sobre las acciones pueden ser tomadas de una red neuronal $h^{(f)}$ donde las probabilidades de la salida son normalizadas al aplicar una función Softmax (8), tal que $\sum a(S_t) = 1$. Este enfoque tiene la siguiente ventaja, si la política optima π^* que busca aprender es determinística entonces la política π aprendida se comportara de forma determinística de igual manera, siempre y cuando la parametrización tenga la capacidad de representar este comportamiento. El comportamiento determinístico en la política se logra debido a que la probabilidad de elegir la acción optima será infinitamente más grande que cualquiera de las acciones subóptimas disponibles (Sutton y Barto,2018):

$$\pi(A_t|S_t, \theta_\pi) \coloneqq softmax\left(h^{(f)}(S_t, \theta_\pi)\right)_{A_t} \tag{19}$$

Donde:

- π define la política,

- $h^{(f)}$ proporciona la salida de la red neuronal sin normalizar,

- A_t es una acción que pertenece al conjunto de todas las acciones validas en el estado S_t,
- s establece el estado actual del ambiente,
- θ son los parámetros de la red neuronal.

3.11.2 Reforzamiento (REINFORCE)

El reforzamiento es una las técnicas destacadas que cuenta con una prueba teórica en Williams (1992), esta es una versión simplificada del descenso de política de gradientes que se enfoca en la acción A_t en el tiempo t, en vez de considerar todas las acciones validas a en tiempo t. La función que describe este método es:

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \tag{20}$$

Donde:

- A_t es la acción tomada en el tiempo t,
- S_t define el estado en el tiempo t,
- θ_t establecen los parámetros de política en el tiempo t,
- G_t es el beneficio obtenido al tiempo t.

La interpretación del reforzamiento en la ecuación (20) es: "El vector es la dirección en la cual el espacio vectorial incrementa la probabilidad de repetir la acción A_t en visitas futuras del estado S_t" (Sutton y Barto, 2018). La división de $\nabla_\theta \pi(A_t|S_t, \theta_t)$ y $\pi(A_t|S_t, \theta_t)$ limita el efecto de las acciones con altas probabilidades de ser seleccionadas, previniendo que exploten sin control y reduce las ventajas con respecto a las acciones que tiene poca probabilidad de ser seleccionadas cuando se visita el estado S_t. En otras palabras, el gradiente será inversamente proporcional a la probabilidad de la acción A_t.

La ecuación (20) es equivalente a:

$$\theta_{t+1} = \theta_t + \eta G_t \nabla_\theta \, ln\big(\pi(A_t|S_t, \theta_t)\big) \tag{21}$$

Un problema del reforzamiento que dificulta su convergencia cosiste en una alta varianza del beneficio G_t durante el aprendizaje. Por este motivo, se agrega una función base b para reducir la variación con la única restricción de que b dependa únicamente del estado S_t. Entonces, la función base $b(S_t)$ reduce la varianza y tiene un efecto nulo en el valor esperado de la función estado-valor:

$$\theta_{t+1} = \theta_t + \eta((G_t - b(S_t) \nabla_\theta \, ln(\pi(A_t|S_t, \theta_t)) \tag{22}$$

Diferentes estrategias existen para definir esta función base, varias de ellas dependen del problema que se afronte, incluso se han desarrollado estrategias como utilizar un decaimiento exponencial. Existe una alternativa general bien recibida, este enfoque utiliza una función parametrizada para estimar la función estado-valor, la utilización de otra función parametrizada otorga a la función base un dinamismo para adaptarse rápidamente a cualquier cambio en la entrada, estima la función estado-valor V desconocida a priori. Esta función necesita ser entrenada por lo que se entrena en paralelo junto con la política, de esta manera, se puede aproximar por medio de muestreo, por esta cuestión se tiene un método Monte Carlo:

$$\hat{V}(S_t, \theta_v) \approx V(S_t) \tag{23}$$

El pseudocódigo del algoritmo de reforzamiento se presenta a continuación:

Algoritmo 1 Reforzamiento (REINFORCE)

Entrada (

- Política $\pi(a|s, \theta_\pi)$
- Función parametrizada de aproximación $\hat{V}(S, \theta_v)$
- Valores del paso de aprendizaje: α_π, $\alpha_v > 0$
- Cantidad de iteraciones k

):

Inicializar los pesos θ_π y θ_v

for 0 **to** k:

dict <- Generador de episodios siguiendo $\pi(.|.,\theta)$

/ comentario: el diccionario "dict" contiene: los estados: $S_0, S_2, .. S_{T-1}$ y las recompensas: $R_1, R_2, \dots, R_T$, correspondientes hasta el estado T*

*/

for $i \leftarrow 0$ **to** T:

$$G = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

$$\delta = G - \hat{v}(S_t, \theta_v)$$

$$\theta_v \leftarrow \theta_v + \eta_v \delta \nabla_v \hat{v}(S_t, \theta_v)$$

$$\theta_\pi \leftarrow \theta_\pi + \eta_\pi \delta \nabla_\pi \pi(S_t, \theta_\pi)$$

Fin

En este caso, el pseudocódigo del algoritmo cuenta con la constante $\gamma \in (0,1]$ para determinar el nivel de influencia de los estados anteriores sobre el estado actual, el caso general $\gamma = 1$.

Capítulo 4 Algoritmos evolutivos

Los algoritmos evolutivos (EA, por sus siglas en inglés) fueron propuestos por Rosenberg en los años 60 para resolver problemas multi objetivos (MOP, del inglés Multi-Objective Problem) pero su aplicación se llevó a cabo hasta mediados de los años 80 con los trabajos de Schaffer, desde su aparición han estado en constante desarrollo en busca de nuevas y mejores estrategias (Coello Coello, 1999).

La ventaja más importante de los EA radica en que no se necesita conocimiento específico del problema, por ejemplo, dinámicas internas, comportamiento de las funciones y relación entre funciones. Esta ventaja permite a los EA comportarse como una metaheurística para resolver problemas computacionales generales. El único requisito necesario es poder evaluar una solución dada dado que las soluciones aprovechan procesos estocásticos para ser producidas (Smith, 1999). Esta situación evita las complicaciones de otros métodos, por ejemplo, los métodos matemáticos (Coello Coello, 1999).

Los procesos utilizados en los EA tienen como inspiración la naturaleza por lo que se les suele denominar bio inspirados. Una estructura denominada *individuo* codifica una solución del problema, esta codificación es correspondiente a un genotipo biológico. En computación, los *genotipos* son representados por medio de cadenas de texto y su decodificación expresa un *fenotipo*. Cada genotipo está compuesto de uno o múltiples cromosomas y estos a su vez, están compuesto de múltiples genes. La posición de los genes se conoce como *locus*. Los genes pueden tomar ciertos valores que son llamados *alelos.* El conjunto de individuos es llamado *población*. La Figura 9 presenta un ejemplo de la estructura de los componentes de los EA (Coello Coello et al., 2007).

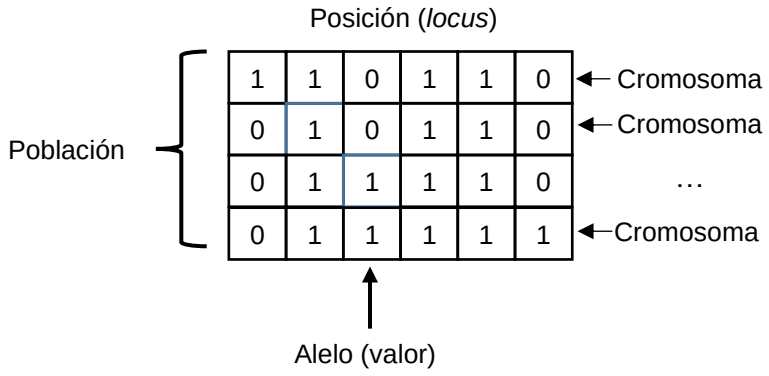


Figura 9. Estructura generalizada de una solución en un algoritmo evolutivo.

Los EA definen una función de aptitud o *fitness* y una función objetivo, la primera es del dominio del algoritmo y la segunda del dominio del problema. Sin embargo, suelen ser idénticas en la práctica y son utilizadas para el mismo propósito.

4.1 Operadores evolutivos

Así como la estructura, los operadores utilizados en los EA también son bio inspirados y reciben el nombre de operadores evolutivos. Los operadores evolutivos de uso más generalizado en la literatura son: mutación, recombinación y selección.

La codificación de los individuos y las consideraciones de la implementación de los operadores depende de la naturaleza del problema, existen otros operadores que son variaciones o generalizaciones que se mencionaran a continuación. Sin embargo, una inadecuada representación del individuo o una mala selección de operadores pueden perjudicar el proceso evolutivo, lo cual se representa en una baja calidad de las soluciones encontradas.

4.1.1 Mutación

El operador de mutación permite a un individuo presentar un cambio de carácter aleatorio en uno o más alelos, estos cambios ayudan a explorar diversas soluciones que no serían contempladas de otra manera, incluso soluciones que no forman parte de la población inicial pueden ser generadas de esta manera. Los operadores de mutación deben considerar la estructura del individuo porque las mutaciones pueden generar individuos no válidos para el dominio del problema. Dos soluciones posibles a este problema son restringir los individuos a solo elementos validos o castigar a los individuos no validos por medio de la función de aptitud. Existen múltiples operadores de mutación, algunos comúnmente utilizados son:

- Mutación volteo de bit (*bitwise mutation*) este operador cambia algún valor de la cadena por el otro elemento binario correspondiente.

- Mutación de intercambio (*swap mutation*) intercambia los valores (alelos) de dos posiciones al azar de un individuo.

- Mutación revólver (*scramble mutation*) selecciona un subconjunto de elementos del cromosoma y los revuelve aleatoriamente.

- Mutación de inversión (*inversión mutation*) selecciona un subconjunto continuo de alelos de un individuo e invierte su orden.

4.1.2 Recombinación

También conocido como cruzamiento, la recombinación permite generar nuevos individuos (descendencia) a partir de las características de dos individuos (padres) en la población actual, en el sentido más sencillo todos los individuos tienen la misma probabilidad de ser padre, sin embargo, este mecanismo no permitía explotar otras soluciones, por este motivo se acompaña de otro operador llamado selección, el cual es presentado más adelante. Existen diversas maneras de realizar una recombinación entre las que tenemos:

- **Cruce de un solo punto,** opera sobre individuos binarios, selecciona un punto al azar en la cadena binaria y corta a ambos padres en ese punto, para después recombinarlos con una pieza del otro individuo. De esta operación se puede obtener uno o dos hijos, dependerá de los detalles de la implementación.

- **Cruzamiento de l puntos**, sigue con la misma dinámica del cruzamiento de un punto, selecciona l puntos de corte al azar en los padres, y se efectúa la misma operación para cada corte seleccionado que se haría para el cruce de un punto.

- **Cruzamiento uniforme,** típicamente cada posición de los elementos que conforma a los individuos tiene la misma probabilidad de ser tomado para formar a un nuevo individuo, además este puede ser visto como el cruzamiento de l puntos cuando es llevado al máximo cantidad de puntos y se aplica una probabilidad para realizar la recombinación.

- **Cruzamiento segmentado**, similar a los anteriores pero la cantidad de puntos de cruzamiento varían entorno a un valor esperado; cada vez que se aplica el operador.

- **Cruzamiento mezclado,** se puede aplicar a cualquier cruzamiento de $l \geq 2$ puntos, consiste en revolver las posiciones (locus) de los genes antes de aplicar el cruzamiento, y después de la recombinación, tanto en hijos como padres regresar la posición de los genes a sus posiciones originales. Tiene como fin reducir el sesgo que se crea al elegir un punto de cruce.

4.1.3 Selección

El operador de selección define los individuos que permanecerán en la población actual para formar la siguiente generación de individuos (descendencia), por lo tanto, los elementos no seleccionados son eliminados en la siguiente generación.

Una forma básica y simple de elegir consiste en una selección aleatoria donde los individuos que tengan una aptitud mayor tienen una mayor probabilidad de ser elegidos, es decir, los individuos con mayor aptitud aportaran más descendientes a la siguiente generación. La importancia de la selección radica principalmente en determinar el balance que existe entre exploración y explotación de las soluciones. De esta manera, existe una gran cantidad de variaciones para este operador, algunos ejemplos son:

- **Torneo** (*Tournament*): Selecciona una cantidad de individuos de la población, y de estos se selecciona a los mejores para generar a la siguiente generación, de estos el más común es el torneo binario (Bäck, 1996).

- **Rango** (Ranking): Asigna la probabilidad de selección solo con base al rango que es relativo del individuo ignorando el valor absoluto de aptitud (fitness) o puede ser una derivación de la aptitud (fitness) y otras implicaciones derivadas de las relaciones entre individuos como lo es caso de los algoritmos: NSGA-II, MOCell.

- **Ruleta**: Asigna al individuo la probabilidad de selección en proporción a la razón obtenida del valor de aptitud del individuo (o ranking) y la media aritmética de la población.

- **Selección** $(\mu + \lambda)$ y (μ, λ): Sea μ la cantidad de padres y λ la cantidad de descendientes para la siguiente generación. $(\mu + \lambda)$ selecciona los μ mejores individuos de los padres y los hijos. (μ, λ) toma a los mejores μ individuos solo de la población de descendientes.

La población de un EA es constante en su mayoría, por ejemplo, un EA con una población inicial de 100 individuos finaliza con el mismo número de individuos. Por lo tanto, el mecanismo de selección de supervivientes determina los elementos que conformaran la siguiente generación y elimina a los no elegidos. La selección de los supervivientes suele ser determinística, en general existen dos métodos: basados en la función de aptitud (los más aptos sobreviven) y basada en edad (la descendencia remplaza a los padres).

A la supervivencia del mejor individuo se le llama elitismo, cuando se aplica la selección de supervivencia basado en aptitud y edad. El elitismo es necesario en un EA porque su ausencia no se garantiza la convergencia del algoritmo (Rudolph, 1994).

4.2 Optimalidad de Pareto

Las definiciones presentadas con respecto a soluciones de Pareto son de carácter general y se han adecuado para ser presentados en el área de EA. Tales adecuaciones presentadas originalmente por Veldhuizen (1999) eliminan la ambigüedad en el manejo de la terminología.

En cualquier generación de un EA puede existir un conjunto de soluciones de Pareto P que está denotada por $P_{actual}(t)$ donde t es la generación a la cual pertenece este conjunto. Esta denotación evita una contradicción con la definición original debido a que solo existe un conjunto de Pareto y no múltiples cómo se maneja en la literatura de los EA.

Distintas implementaciones de EA utilizan un archivo para mantener las mejores soluciones de Pareto encontradas hasta el momento t, durante distintas generaciones. El archivo $P_{conocidos}(t)$ puede ser utilizado de diferentes maneras dependiendo la implementación (Veldhuizen, 1999). Al terminar la ejecución del algoritmo, se espera que $P_{conocidos}(t)$ contenga el conjunto de soluciones Pareto óptimas del problema, en la mayoría de los casos. Además, debido a que el conjunto de Pareto óptimo $\mathcal{P}^*$ no es conocido a priori, es necesario definir $P_{verdadero}$ como un subconjunto apto para el dominio computacional.

4.3 Algoritmos Genéticos

Los algoritmos genéticos (GA, por sus siglas en inglés) son sin duda los más conocidos y utilizados dentro de los enfoques de EA. En un inicio, no fueron considerados como algoritmos de optimización, sin embargo su capacidad para resolver problemas y los resultados han mostrado lo contrario a pesar de sus detractores (Bäck, 1996).

Los GA utilizan poblaciones de individuos las cuales son idóneas para la optimización multi objetivo. Las poblaciones están constituidas de múltiples soluciones por lo que solo se necesita hacer una sola ejecución del algoritmo para obtener el frente Pareto, a diferencia de otros métodos que utilizan varias ejecuciones para encontrar todas las soluciones que lo conforman. Otra ventaja expuesta por Coello Coello (1999) es que "*los AG son menos suseptibles a la forma o la continuidad del frente de Pareto, mientras estos problemas son una preocupación real para tecnicas de programación matematica"*.

Para encontrar el conjunto de soluciones, los GA utilizan distintos enfoques, entre los más conocidos se encuentran:

- Método de agregación, combina los objetivos para producir un único valor real.
- Enfoque no basado en el óptimo de Pareto, modela los objetivos como un vector.
- Enfoque basado en el óptimo de Pareto, mantiene los objetivos como un vector.

El método más utilizado dentro del área de GA es el basado en la optimalidad de Pareto. Esta clasificación es compatible por la presentada por Cohon y Marks (1975) para la clasificación de problemas multi objetivo, existen diferentes clasificaciones pero cada una está diseñada para resaltar algunos elementos de los algoritmos utilizados.

4.3.1 Método de agregación

El método de agregación usualmente combina múltiples objetivos en un nuevo y único objetivo a optimizar tras una serie de operaciones. Una manera natural es usar una combinación lineal de los objetivos (suma

ponderada) donde se toma el vector de funciones objetivo junto con escalares como coeficientes de los objetivos; los escalares deben de estar pensados de manera que una función no domine a otra.

Este es uno de los métodos más eficientes que se pueden emplear en un GA debido a que no se necesita interacción con el usuario para tomar decisiones. Además, si se logra optimizar la función de aptitud, entonces el resultado será al menos una solución sub optima en la mayoría de los casos (Coello Coello, 1999). Sin embargo, se requiere cierto conocimiento del comportamiento de las funciones a optimizar para que el método funcione adecuadamente. Una desventaja de la implementación de un GA con una función de agregación es la generación de una única solución al final de la ejecución.

Otros métodos en EA basados en métodos de agregación son: Goal programming (Charnes y Cooper, 1957), goal attainment, ε constraint, etc.

4.3.2 Enfoque no basado en el óptimo de Pareto

Estas técnicas definen políticas en el manejo de los objetivos o la población. VEGA (del inglés, Vector Evaluated Genetic Algorithms) (Schaffer, 1985) es un ejemplo de este enfoque donde el operador de selección se modifica para dividir la población en k subpoblaciones y cada población se especializa en mejorar un objetivo en específico. Estas subpoblaciones son mezcladas y los pasos siguientes son llevados a cabo como cualquier otro EA. Un problema frecuente de este tipo de algoritmos es la hiper especialización en los objetivos, por lo tanto, se puede encontrar un frente de Pareto muy pobre en diversidad.

Otras técnicas que no toman en cuenta a Pareto optimo son: Orden lexicográfico, teoría de juegos, uso de género (Allenson, 1992), Enfoque Min-Max, etc.

4.3.3 Enfoque basado en él óptimo de Pareto

Este enfoque utiliza una clasificación de no dominación y selección con el objetivo de que la población se aproxime al frente de Pareto (Goldberg, 1989). Inicialmente, los individuos no dominados de la población son clasificados con el mayor valor y separados de la población. El proceso se repite con la población restante para encontrar y clasificar las soluciones no dominadas hasta que todas las soluciones estén

correctamente clasificadas. Además, se utilizan técnicas adicionales para evitar que todas las poblaciones solo pertenezcan a una sola sección del frente de Pareto (diversificación).

Estos algoritmos establecen metas específicas como:

- Meta 1: Preservar las soluciones no dominadas.
- Meta 2: Progresar o guiar $P_{actual} \rightarrow P_{verdadero}$.
- Meta 3: Generar y mantener diversidad de puntos del frente de Pareto.
- Meta 4: Proveer al usuario tomador de decisiones con un limitado pero variado de $P_{conocidos}$ al terminar la ejecución.

Existen dos generaciones de este tipo de GA, la primera sin elitismo y la segunda con él. La segunda generación es la más utilizada debido a que el elitismo es parte fundamental para la convergencia.

Los GA para resolución de problemas multi objetivo se centran en obtener soluciones lo más cercanas posibles a $P_{verdadero}$. Además, los GA tienen una buena diversidad de soluciones distribuidas a lo ancho de todo el frente de Pareto, lo suficientemente grande para que el tomador de decisiones le sea de utilidad. Estos dos puntos son logrados por la clasificación basada en rango y preservación de la diversidad.

4.3.4 Clasificación basada en rango

El operador de dominancia de Pareto es un operador binario con la propiedad de orden parcial estricto (irreflexivo, transitivo y asimétrico). Un punto en el espacio de solución puede ser dominado, no dominado o incomparable. Algunas técnicas utilizan el operador de dominancia antes del operador de selección, por ejemplo:

- Clasificación por dominancia, define el número de individuos que dominan a un individuo en particular.

- Dominancia por conteo, determina el número de individuos dominados por un individuo en particular.

- Dominancia por profundidad, establece el frente de Pareto al cual pertenece un individuo en particular.

La Figura 10 muestra un ejemplo de la clasificación por dominancia, donde existen tres clasificaciones los no dominados, los dominados por 2 soluciones y los que son dominados por 3.

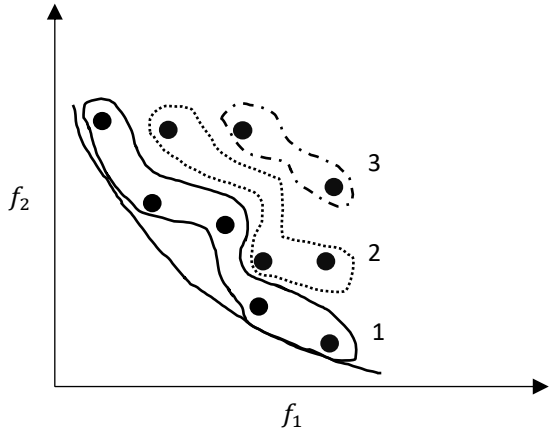


Figura 10. La clasificación por dominancia agrupa a los individuos por el número de individuos dominados.

4.3.5 Preservación de la diversidad

Una característica de los algoritmos basados en dominancia de Pareto es la estrategia de búsqueda de soluciones, su objetivo es preservar la diversidad entre las soluciones encontradas por lo que recompensa ciertas soluciones en la función de aptitud. De esta manera, se pueden obtener soluciones dispersas a través de todo el frente de Pareto. Algunos ejemplos de estas estrategias son:

- Búsqueda sesgada. Un vector de pesos sesga las nuevas soluciones que se alejan sus vecinos.

- Enfoque de nicho. Se reduce la función de aptitud del individuo proporcionalmente por la cantidad de vecinos que tenga. Los vecinos se pueden determinar de distintas maneras, puede ser por distancia cómo L_2 (*kernel approach*), dividir el espacio solución en una rejilla donde los vecinos

son todas las soluciones que estén en la misma rejilla (*grid*), o estimar el volumen del hiper rectángulo formado por las soluciones vecinas (*Nearest neigboard approach*).

- Distancia de hacinamiento. Similar al enfoque de nicho pero más eficiente, esta clase de implementación es utilizada por algoritmos como el NSGA-II.

- Restricción en la recombinación. Evita combinar individuos cercanos.

- Dominancia relajada. Relaja el concepto de dominancia para permitir que más individuos no sean dominados en uno o varios objetivos agregando una tolerancia ε a cada uno.

4.3.6 Algoritmo genético de ordenamiento no dominado (NSGA-II)

El algoritmo genético de ordenamiento no dominado (NSGA-II) (Deb et al., 2002) es una mejora de su predecesor NSGA-I. Este algoritmo reduce el tiempo computacional necesario para ordenar las soluciones no dominadas en $O(MN^2)$ donde M es la cantidad de objetivos y N es el tamaño de la población. Además, usa elitismo e introduce un mecanismo para preservar la diversidad del frente de Pareto que no requiere de parámetros.

NSGA-II opera como un GA tradicional, pero genera una descendencia Q_t con la misma carnalidad que la población K_t. Posteriormente, todos los individuos se unen y los elegidos forman parte de la siguiente generación K_{t+1}. La selección depende de un ordenamiento parcial donde se escoge las soluciones clasificadas en el menor rango. Sí dos soluciones pertenecen a la misma clasificación entonces se prefiere la solución que se encuentre en el área con menos soluciones. Este procedimiento se ve reflejado en la Figura 11.

El proceso se repite hasta alcanzar algún criterio de terminación. De esta manera, las soluciones en $P_{verdadero}$ cuentan con una buena diversidad a lo largo del frente de Pareto.

Para calcular la distancia de hacinamiento entre las soluciones, la población se ordena con respecto al objetivo f_i y posteriormente se suma a la distancia total al promedio normalizado de la solución que tenga por encima y por debajo del valor objetivo f_i. Este proceso se repite para todos los objetivos.

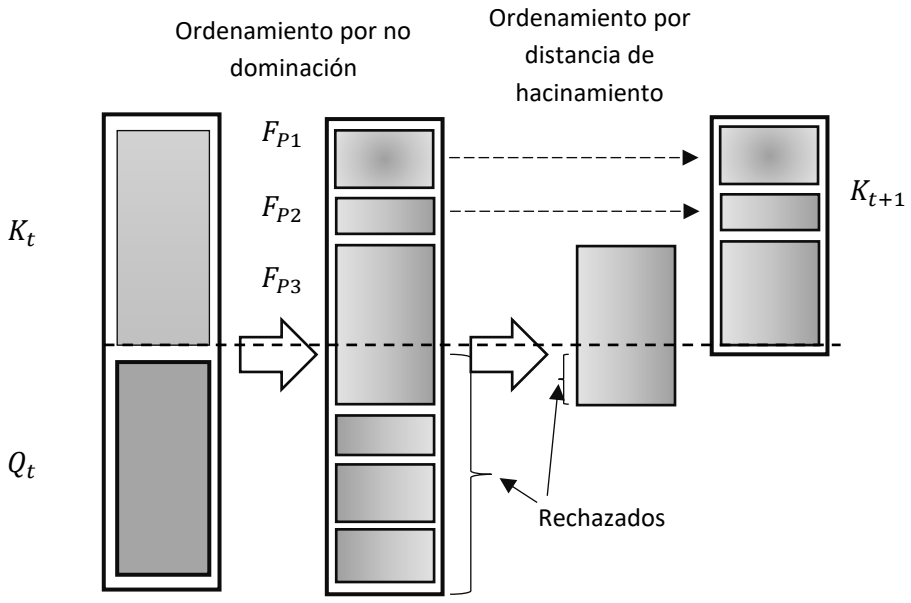


Figura 11. Procedimiento general del NSGA-II.

4.3.7 Algoritmos genéticos celulares

Los algoritmos genéticos celulares hacen un uso no tradicional de la población donde todas las soluciones se encuentran en diferentes archivos o no se tiene acceso a su totalidad. Existe dos variantes principales de estos algoritmos: la población distribuida y la población celular. La descripción de este texto se enfoca en la segunda variante.

En un algoritmo celular, los individuos de la población se organizan en estructuras de diversas formas: rejilla simple, una rejilla toroidal o alguna otra estructura que pueda ser beneficiosa. De esta manera, cada individuo solo tiene permitido interactuar con vecinos pertenecientes al mismo vecindario. Los vecindarios normalmente se agrupan en estructuras de rejilla con variaciones, las mayores diferencias son la forma y el número de vecinos con los que puede interactuar una solución. El vecindario C9 es uno de los más utilizados donde 9 solucione se agrupan en una estructura de 3×3 elementos.

Algunos de los algoritmos celulares más conocidos para optimización multi objetivo son:

- Algoritmo genético celular multiobjetivo - CMOGA (Alba et al., 2007).

- Algoritmo evolutivo celular multiobjetivo - MOCell (Nebro et al., 2014).

La diferencia entre CMOGA y MOCell es que este último utiliza una lista externa para almacenar los elementos no dominados.

4.3.7.1 Algoritmo evolutivo celular multiobjetivo (MOCell)

MOCell organiza la población en una rejilla toroidal con un vecindario C9. Su mayor contribución es introducir una lista finita de elementos no dominados para retroalimentar nueva generación. El ciclo principal aplica operaciones de mutación y recombinación a los elementos por vecindarios. Posteriormente, los hijos resultantes del vecindario son evaluados con la función de aptitud, los hijos resultantes reemplazan a los padres si los dominan y son almacenados en el archivo auxiliar sí no son dominados por sus elementos (se utiliza el mismo criterio de dominancia que el NSGA-II). Al final de cada ciclo, después que todos elementos/vecindarios hayan pasado por este proceso, las soluciones no dominadas del archivo son reintegradas de forma aleatoria a la población, el ciclo mencionado anteriormente se repite hasta que se cumpla con el criterio de paro. La Figura 12 representa el ciclo del algoritmo MOCell.

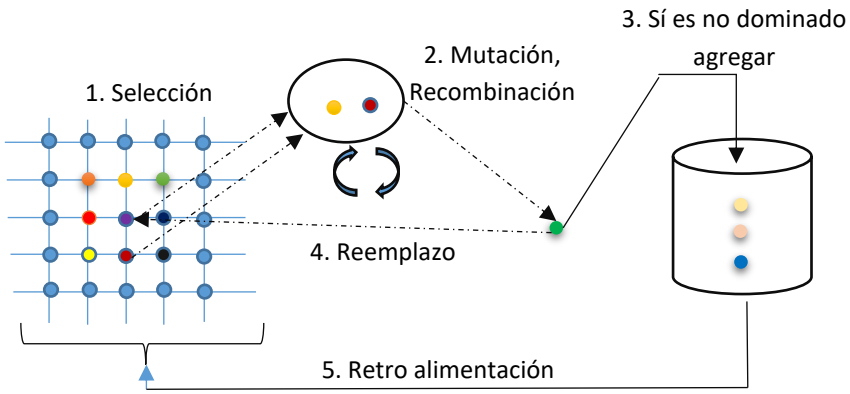


Figura 12. Procedimiento general de MOCell.

4.3.8 Framework JMetalPy

JMetalPy es un framework en Python basado en JMetal de JAVA para optimización muti objetivo, este framework aprovechar las ventajas propias del lenguaje: prototipado rápido y el uso de librerías de código libre. JMetalPy cuenta con múltiples implementaciones de metaheurísticas como: NSGA-II, GDE3, SMPSO, OMOPSO y MOEA; así como una versión dinámica del NSGA-II y SMPSO. Además, incluye indicadores de calidad como hipervolumen, épsilon aditiva y distancia generacional invertida.

Otras herramientas adicionales de JMetalPy consideran la visualización y la implementación de algoritmos con Apache Spark y Dask.

Capítulo 5 Metodología

En este capítulo, se define el problema de optimización de la confiabilidad y redundancia en un sistema de almacenamiento basado en AR-RRNS con una red neuronal apuntadora. Se precisan algunas consideraciones de su implementación, librerías y la metodología de la evaluación experimental.

5.1 Metodología

La metodología utilizada para abordar el problema consistió de cuatro fases principales, cada fase consta de diferentes actividades descritas a continuación:

Fase 1: El uso de redes neuronales para resolver problemas de optimización es un tema relativamente reciente, para entender sus alcances y limitaciones fue necesario realizar una exploración de sus aplicaciones a la fecha.

- Revisión de la literatura: Se realizó una investigación de los últimos avances en el área y se identificaron los trabajos relevantes para la investigación.

- Análisis de arquitecturas y tipos de redes neuronales: A la par de la revisión de la literatura, se estudiaron diferentes arquitecturas aplicadas para la resolución del problema o estrategias empleadas en las redes neuronales para abordar el problema de interés.

Fase 2: La información recolectada en la fase anterior sirvió para establecer las especificaciones de la red neuronal y comenzar con su desarrollo.

- Configuración: Se seleccionó una configuración de red neuronal en específico para su implementación y se definió el método de ajustar sus pesos (entrenamiento). La elección del método de entrenamiento está basada fuertemente en el tipo de datos y problema que se aborda.

- Selección de los parámetros a evaluar: Se establecieron los parámetros a evaluar en la implementación del modelo.

Fase 3: La implementación del modelo y su verificación se desarrollaron durante esta fase, el objetivo es tener certeza sobre el correcto funcionamiento de la red.

- Implementación del sistema. Se desarrolló la implementación de las funciones internas de la red neuronal, utilizando estándares de programación, bibliotecas públicas y herramientas libres y ampliamente utilizadas en la comunidad científica.

- Implementación de otros algoritmos. Se desarrollaron e implementaron otras estrategias para comparar la eficiencia de la red neuronal.

Fase 4: Los resultados obtenidos son analizados después del entrenamiento y las pruebas pertinentes.

- Análisis de resultados. Se comparar el desempeño del modelo con otros algoritmos para entender su comportamiento y la capacidad de resolver problemas de este tipo.

- Conclusiones. Se describen los mayores hallazgos encontrados en la sección de evaluación.

5.2 Definición del problema con enfoque en redes neuronales

En esta sección se describe el problema de optimización AR-RRNS, para ser resuelto por medio de una red apuntadora (Ptr-Net) con enfoques mono objetivo y multi objetivo. Se platean los distintos escenarios, consideraciones en la codificación y tratamiento de los datos.

5.2.1 Codificación de la de entrada para la red Ptr-Net

La codificación de la entrada consiste en establecer los datos de las nubes, los parámetros de configuración de la red Ptr-Net y los símbolos especiales, esta entrada es común en todas las instancias para otorgar a la red la capacidad de efectuar acciones especiales. Los símbolos especiales son:

- α, define la acción $k + 1$, aumentar redundancia.

- β, establece el término de una secuencia.

El uso de símbolos especiales está condicionado por parámetros de configuración de la Ptr-Net.

La secuencia X de entrada al modelo estará conformado por el conjunto N de nubes y el conjunto de elementos artificiales. Además de los parámetros de configuración que determinan el comportamiento de la Ptr-Net (véase Figura 13).

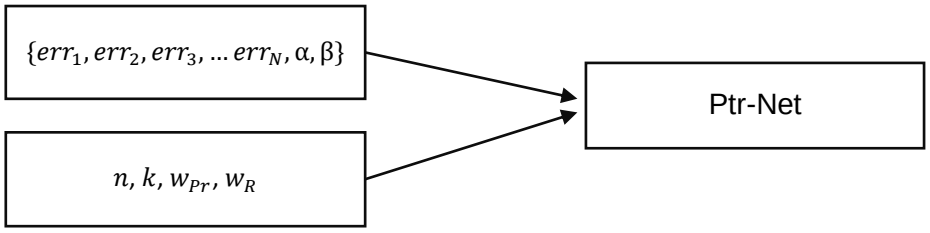


Figura 13. Los parámetros de entrada, elementos del lote y los símbolos artificiales, determinan la inferencia de la red.

5.2.2 Codificación de la solución

La secuencia solución $\pi^{(v)}$ está conformada por un subconjunto de nubes $C^{(s)}$ seleccionadas del conjunto de entrada de N nubes y los elementos α y β. Para la decodificación, la cantidad de nubes en $\pi^{(v)}$ este dado por $n_s = \left|C^{(s)}\right|$ y la cantidad de redundancia por $k_s = \sum_{i=0}^{n} r_i$, $r_i = \begin{cases} 1, if \pi_i^{(v)} = \alpha \\ 0, if \pi_i^{(v)} \neq \alpha \end{cases}$, véase Figura 14.

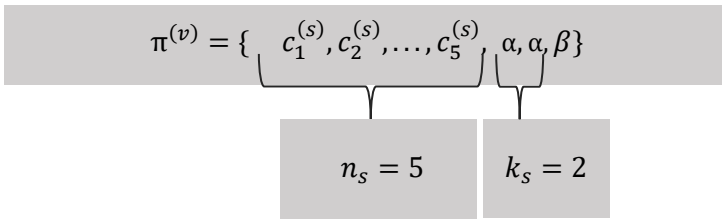


Figura 14. Decodificación de la solución.

5.2.3 Funciones objetivo y evaluación de la solución

En el planteamiento del problema se busca minimizar las funciones definidas por la probabilidad de perdida de información y redundancia. La probabilidad de pérdida de información denotada por $Pr(k,n)$ o Pr para simplificar, está dada por:

$$Pr(k,n) = \sum_{A \in \widehat{F}_{n-k+1}}^{n} \prod_{j \in A} err_j \prod_{j^c \in A^c} \left(1 - err_{j^c}\right) \tag{24}$$

Donde:

- $\widehat{F}_{n-k+1}$ define los subconjuntos de tamaño $n-k+1$ del conjunto $C^{(s)}$, cada subconjunto de tamaño $n-k+1$ representa una posibilidad de pérdida de información,
- err_j es la probabilidad de error de la nube $c_j^{(s)} \in C^{(s)}$,
- A establece un elemento del conjunto $\widehat{F}_{n-k+1}$,
- A^c es el complemento del conjunto A.

La ecuación (24) no es compatible con el método de agregación por lo que necesita ser adaptada, en vez de utilizar solo el valor de $Pr(k,n)$ se le aplicó una transformación basada en el logaritmo natural,

$$Pr_{ln}. = ln(Pr) \tag{25}$$

Para obtener la redundancia del sistema se tiene que:

$$R = \frac{D_E}{D} \tag{26}$$

Donde:

- R es la razón de la información codificada entre tamaño original de la información,
- D_E es el tamaño de la información codificada,

- D es el tamaño original de la información.

Las funciones objetivo (25) y (26) están en conflicto porque el mejorar alguna de las dos funciones actúan en detrimento de la otra función. La secuencia $\pi^{(v)}$ generada por la Ptr-Net es decodificada para obtener los respectivos valores de las funciones presentadas. Posteriormente, ambos valores son normalizados con la ecuación (32) para que se encuentren en el mismo rango de valores. Finalmente, se obtiene $F = \left[Pr_{n\_ln}, R_n\right]$, la sección 5.2.6 proporciona más información sobre los cambios a las funciones.

Los valores (k, n) deben cumplir con las restricciones del problema AR-RRNS para garantizar validez en las soluciones, las restricciones son:

1) $n \geq 2$, al menos dos nubes deben ser seleccionadas.

2) $k \geq 2$, para poder recuperar (reconstruir) la información.

3) $k \leq n$, la cantidad de redundancia del sistema debe ser menor que la cantidad n de partes en las que se divide la información.

Las restricciones son integradas en la Ptr-Net por lo que secuencias $\pi^{(v)}$ generadas son válidas.

5.2.4 Estructura Ptr-Net y consideraciones

En esta subsección, se presentan las adecuaciones de la red siguiendo las recomendaciones establecidas por Bello et al. (2016). La red apuntadora se compone de dos estructuras integradas: la arquitectura codificación-decodificación y el mecanismo atención que forma parte del decodificador.

El codificador utiliza una estructura LSTM bidireccional y el decodificador utiliza una LSTM simple donde ambas LSTM no comparten pesos. La implementación del mecanismo de atención se compone de tensores cuadrados y un vector columna, los detalles de ambos componentes se presentan en los siguientes párrafos, junto con la integración de las restricciones del problema.

El codificador procesa toda la secuencia de entrada para entregar una representación intermedia y el vector de contexto es el último estado interno de la LSTM. Posteriormente, el decodificador entra en un

bucle para generar la solución $v^{(\pi)}$. Como parte de este bucle, el mecanismo de atención (14) genera la distribución de los elementos de la entrada, pero con las adaptaciones propuestas en Bello et al. (2016) para integrar las restricciones del problema.

El mecanismo apuntador es conformado por una serie de matrices de atención (valores que parametrizan al mecanismo) $W_{ref}, W_q \in \mathbb{R}^{d\times d}$ y un vector de atención $v \in \mathbb{R}^d$, similar a (14) pero con una condición adicional para determina si un elemento es válido en la secuencia de salida, de manera se tiene que:

$$u_j^i = \begin{cases} v^T \cdot tanh\big(W_{ref} \cdot r_j + W_q \cdot q_i\big), j \in (1,2,\dots,n) & Sí\ se\ cumplen\ las\ restricciones \\ -\infty, & En\ caso\ contrario \end{cases} \quad (27)$$

Las restricciones aplicadas para la generación de la secuencia $\pi^{(v)}$ dependerán de los parámetros de configuración de la Ptr-Net. Específicamente si se designa algún valor para k y/o n tal que se genere una restricción de igualdad. Los casos que se contempla son los siguientes:

1) El sistema tiene libertad de escoger n y k cuando las variables n y k no son definidas a priori:

   a. Sí el token u_j^i no es de algún elemento que se encuentre en $\pi^{(v)}$ y el elemento especial de fin de secuencia β no ha sido seleccionado previamente;

   b. Sí se mantiene la condición $k_s \leq n_s$, y el elemento β no ha sido seleccionado previamente;

   c. Especial para el u_j^i correspondiente a β, si $|C_s| \geq 2$ y $k_s \geq 2$.

2) n ha sido definido como $2 \leq n \leq |N|$:

   a. Sí el token u_j^i no es de algún elemento que se encuentre C_s y $|C_s| \leq n$;

   b. Especial para el u_j^i correspondiente a α, satisface la condición $k_s \leq n$, y el elemento β no ha sido seleccionado previamente.

   c. Especial para el u_j^i correspondiente a β, sí $|C_s| = n$ y $k_s \geq 2$.

3) k ha sido define a priori como $2 \leq k \leq |N|$:

   a. Sí el token u_j^i no es de algún elemento que se encuentre C_s y β no ha sido seleccionado previamente;

   b. Especial para el u_j^i correspondiente a β, sí $|C_s| \geq k$.

El vector u^i actúa como una máscara sobre los elementos por lo que es necesario aplicar la función SoftMax (8) para obtener su probabilidad, los elementos que no cumplan con las restricciones tendrán asignada una probabilidad de 0, ambos procesos se identificado con la función A:

$$A(ref, q_i; W_{ref}, W_q, v) \coloneqq softmax(u^i) \tag{28}$$

Donde:

- ref son los elementos de referencia de los cuales se pondera su contribución para generar elemento u_j^i,
- q_i es la consulta actual,
- W_{ref}, W_q y v los parámetros del sistema apuntador.

En cada iteración j, la Ptr-Net selecciona un elemento del conjunto de entrada compuesto por las nubes y elementos especiales de acuerdo con las restricciones del sistema. Entonces, la probabilidad de escoger un elemento $u_j^i \in u^i$ dados la secuencia de entrada X y $\pi^{(v)}$ está definido por:

$$p(\pi(j)|\pi(<j), X) \coloneqq A(enc_{1:n}, dec_j) \tag{29}$$

Donde:

- $\pi(j)$ define el j-ésimo elemento a inferir por la Ptr-Net,
- $\pi(<j)$ son todos los elementos previamente inferidos por la Ptr-Net al j-ésimo elemento,

- X establece la secuencia de entrada de la Ptr-Net,
- $enc_{1:n}$ son las codificaciones de los elementos de la entrada,
- dec_j es la codificación del elemento inferido anteriormente.

Además, el mecanismo de atención (13) permite reformular el elemento q_i a g_l, antes de introducirlo al elemento apuntador con el fin de enriquecer la consulta y sumar parámetros al modelo, porque cuenta con parámetros independiente a (28).

$$p = A\left(ref, q; W_{ref}^{g}, W_{q}^{g}, v^{g}\right) \tag{30}$$

$$\left(ref, q; W_{ref}^{g}, W_{q}^{g}, v^{g}\right) \coloneqq \sum_{i=1}^{k} r_i p_i \tag{31}$$

En la ecuación anterior se puede notar que cada elemento de ref (r_i) es multiplicado por su respectivo nivel de atención y posteriormente reducido a un solo elemento por medio de una suma. Este nuevo elemento es denotado como g_l.

Este proceso puede ser repetido tal que $g_0 = q, g1 = G(ref, g_0), \ldots, g_l = G(ref, g_{l-1})$, sin embargo, no se reporta una mejora considerable en el sistema si es llevado más allá de g_1, de acuerdo con Bello et al. (2016). El vector g_l sustituye a q en el mecanismo apuntador tal que A$\left(ref, g_l; W_{ref}, W_q, v\right)$.

5.2.5 Función de normalización y degradación

En varias secciones se requiere normalizar los datos por lo que se utiliza la función:

$$normalizado = \frac{valor - minimal}{maximal - minimal} \tag{32}$$

La degradación (Deg) es la razón de lo obtenido por el modelo y el valor mínimal, este será utilizado como un indicador:

$$Deg(valor) = \frac{valor}{optimal} \tag{33}$$

Los valores de Pr_{ln} son negativos por lo que la expresión Deg se tiene que cambiar para que los valores se mantengan relativos al 0, de esta manera se aplica $Deg(valor) = \frac{|optimal|}{|valor|}$.

5.2.6 Función de agrupación – suma ponderada WS

La suma ponderada (WS, weighted sum) es una función de agrupación donde se designan pesos w_i, $\mathrm{i} = 1, .., \mathrm{m}$ a cada m objetivo a optimizar, así el vector de preferencias w determina la importancia de cada objetivo y satisface $\sum_i^m w_i = 1$. Este vector es utilizado para establecer las preferencias de los usuarios

$$WS = Pr_{n\_ln} * w_{pr} + R_n * w_R \tag{34}$$

- w_{pr} es el peso de P_R.
- w_R es el peso que se asigna a la redundancia.
- Pr_{n_ln}: es el valor normalizado de Pr_{ln} por (32).
- R_n: es R normalizado por (32).

Las características de la suma ponderada en conjunto con el método de Reinforce imponen ciertas restricciones, algunas consideraciones para expresadas a continuación con las funciones objetivo.

La función $Pr(k, n)$ entrega valores con magnitudes extremadamente pequeñas y distintas conforme aumenta el valor de n. Para $|N| = 10$ la magnitud mínima es $Pr_{min}\text{Magnitud} \approx 1 \times 10^{-27}$ y el máximo es $Pr_{max}\text{Magnitud} \approx 1 \times 10^{-03}$, y entre estos hay magnitudes como $1 \times 10^{-05}, 1 \times 10^{-10}, 1 \times 10^{-13}, 1 \times 10^{-20}$ y 10^{-24} por exponer algunos.

Contrastando con los valores que toma la función R sus valores son mucho más grandes en magnitud. Para una $|N| = 10$ la función R puede tomar valores como $2.00,\ 2.50, 4.00,\ 6.67$ y 10.00, y una magnitud $R_{min}Magnitud\ \approx 1 \times 10^{0}$ y $R_{max}Magnitud\ \approx 1 \times 10^{1}$

Sí comparamos las funciones por magnitud, una domina naturalmente a la otra en la suma ponderada (función agregada), esto es, R domina naturalmente a Pr, debido a que Pr tiende a aproximarse "rápidamente" a cero a diferencia de R conforme la cantidad de elementos aumenta.

Teniendo en cuenta el comportamiento de Pr se le puede aplicar normalización logarítmica. Una herramienta utilizada en aprendizaje máquina que captura los cambios relativos entre valores, la magnitud del cambio y reduce la varianza en los datos. Posteriormente a ambas funciones objetivo se les ha aplicado una normalización de rango para mapear sus valores al intervalo $[0,1]$.

Una motivación más para realizar esta transformación es debido al uso del aprendizaje por refuerzo, en específico REINFORCE (22). Las recompensas obtenidas y sus magnitudes son importantes, ya que con estas se "recompensa" o "castiga" el desempeño de la Ptr- Net al elegir soluciones en fase de entrenamiento. Recompensas muy pequeñas hace que la red esta "a gusto" con soluciones pobres, debido a que apenas tendrán efecto para motivar a la Ptr-Net en arriesgarse a mejorar la solución encontrada.

Considerando a Pr en este escenario, el aprendizaje sería considerablemente lento. Las recompensas obtenidas son varias magnitudes más pequeñas que la tasa de aprendizaje de la Ptr-Net, el cual comienza en una magnitud de 1×10^{-03}, mucho mayor que varias recompensas obtenidas de Pr.

Esto cuando se habla de optimizar una función, cuando se trata de la función agregada (suma ponderada), son al menos dos funciones, por lo que las funciones que aporten menos a la suma podrían ser ignoradas.

La función Pr no aportaría valor a la suma ponderada, salvo en casos específicos para evitar solo los peores escenarios donde la magnitud es más significativa. Es interesante notar que el efecto es parecido al desvanecimiento del gradiente para algún objetivo, impidiendo que los pesos se ajusten de manera correcta para representar una mejora de los parámetros para ese objetivo.

5.2.7 Entrenando para múltiples subproblemas

Se puede optimizar un subproblema correspondiente a valores de w_R y w_{Pr} entrenando la Ptr-Net. Posteriormente se puede resolver otro subproblema cambiando por alguna cantidad τ los valores de preferencia, tal que $w_R + \tau_1$ y $w_{Pr} + \tau_2$, así como también tomar los parámetros entrenados θ del subproblema anterior (Li et al., 2021)(véase Figura 15).

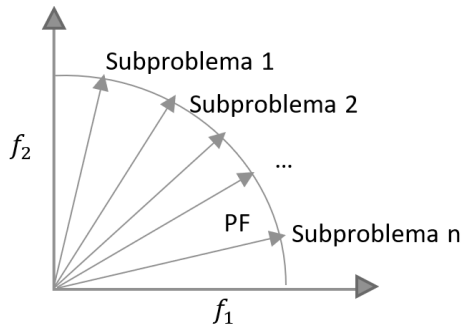


Figura 15. Entrenamiento de subproblemas.

Subproblemas vecinos comparten parámetros similares, así el modelo solo necesita un ajuste fino (fine-tunning) para los subproblemas restantes a entrenar, de esta manera se utiliza menos épocas para converger y se reduce el tiempo total de entrenamientos necesarios para todos los subproblemas a resolver.

5.2.8 Entrenamiento con política de gradientes

El entrenamiento utiliza REINFORCE (22) para ajustar los parámetros internos de modelo, conocidos como pesos y denotados con θ. El objetivo del entrenamiento es el valor esperado de suma ponderada (WS) dado un conjunto de nubes N, presentados en forma de secuencia, formalmente $J(\theta|N) = \mathbb{E}_{\pi\sim p\theta(.|N)} WS\left(\pi^{(v)}\middle|N\right)$. El gradiente de la política de la siguiente manera.

$$\nabla_\theta J(\theta|N) = \mathbb{E}_{\pi\approx_\theta(.|\mathrm{N})}\left[\left(\mathrm{WS}\left(\pi^{(v)}\,|N\right) - \mathrm{b(N)}\right)\nabla_\theta \ln \mathrm{p}_\theta\left(\pi^{(v)}\,\middle|\mathrm{N},\theta\right)\right] \tag{35}$$

Donde:

- $\pi^{(v)}$ define la secuencia solución encontrada por la red apuntadora,
- $WS\left(\pi^{(v)}\,\middle|N\right)$ establece el valor de la suma ponderada de los objetivos obtenida a partir de $\pi^{(v)}$,
- N es el conjunto de nubes,

- $\nabla_{\theta} J(\theta|\mathrm{N})$ representa el vector columna de derivadas parciales de $J(\theta|\mathrm{C})$ con respecto θ dado N,

- $b(N)$ es una función de valor de referencia (Base line) para disminuir la alta varianza durante el entrenamiento. En este caso es al Critic-Net,

- $lnp_{\theta}(\ \ )$ define el logaritmo de las probabilidades bajo la distribución generada bajo los parámetros de θ.

La función base puede ser cualquier función que no dependa de los parámetros de la red principal, en este caso se utilizó otra red neuronal similar en estructura a la Ptr-Net pero con par de capas con neuronas totalmente conectadas a la salida y parametrizada con θ_i (independientes de la primera). Esta red es conocida como *Soft-Critic-Net o Critic-Net* que aprende a predecir el valor esperado de WS dada una entrada N de nubes.

Las redes son entrenadas con GD y la función de pérdida está definida por la función de mínimos cuadrados en el caso de la Critic-Net.

5.3 Configuración experimental

La evaluación experimental se realizó en un equipo de cómputo con las siguientes características: procesador Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 16 Gb de memoria RAM, Windows 10 Home edition y una tarjeta NVIDIA GeForce RTX 2070 Max Q-Design.

La implementación de las estrategias utiliza los siguientes programas y librerías:

- Python 3.7,

- PyTorch 1.7,

- Numpy 1.9,

- JMetalPy 1.5.5,

- Matplotlib.pytplot.

El embedding utiliza una capa convolucional con las siguientes características: tamaño del kernel = 1, stride = 1 y out_channel = 64, Además, el embedding reduce la dimensión original de cada elemento a una dimensión de 64. La dimensión de las capas ocultas en las *LSTM* (codificador y decodificador), el mecanismo apuntador y de atención es $d = 128$.

Los pesos iniciales se generan con una distribución Xavier normal, un optimizador ADAM con valores betas $(0.9,0.999)$ y un factor de aprendizaje inicial de $1e^{-03}$. Un scheduler step de LR actualiza el peso de la época i al multiplicarlo por una constante $gamma = 0.1$ cada 3 épocas. Esta configuración también fue utilizada en el Critic-Net, su función es auxiliar al algoritmo de reforzamiento utilizado en el entrenamiento. La función de perdida para la Critic-NET está definida por el error cuadrático medio (MSE).

La Tabla 1 presenta los valores máximos y mínimos de probabilidad de falla para los proveedores STaaS utilizados en el conjunto de entrenamiento, esta información fue obtenida del servicio CloudHarmony (2015) por medio de un monitorio continuo enfocado en los servicio IaaS, los valores representan el análisis mediante pings constantes durante la puesta en marcha de carga de trabajo en la nube. Este proceso registra los tiempos fuera de servicio del sistema y su probabilidad es calculada por medio de una probabilidad geométrica. Estos datos junto con una distribución uniforme fueron utilizados para generar las instancias de entrenamiento en la etapa de aprendizaje por refuerzo. Los datos generados de los CSPs son normalizados en el rango de [0, 1] con (32), el mayor valor entre las nubes define el máximo en la instancia y la definición del mínimo sigue la misma lógica.

Tabla 1. Valores mínimo y máximo de cada CSP utilizados para la experimentación.

j	Nombre	err_j	
		mínimo	máximo
1	Google Drive	0.00072679	0.00145361
2	OneDrive	0.0006602	0.00132043
3	DropBox	0.00097032	0.00194065
4	Box	0.00179699	0.00359402
5	Egnyte	0.00073249	0.00146503
6	Share File	0.00014269	0.00028542
7	Sales Force	0.00061739	0.0012348
8	Alibaba cloud	0.00079897	0.00138793
9	Amazon Cloud Drive	0.00018227	0.00098241
10	Apple iCloud	0.00015664	0.00097632

El lote de entrenamiento consta de 128 elementos donde cada elemento posee un subconjunto de N, 10 nubes por elemento del lote con sus respectivos parámetros. La posición de las nubes en el conjunto N es permutada para evitar que la red aprenda a ubicar mejores elementos solo por su posición inicial. Los valores de las funciones son normalizados. Una vez entrenada la red, se utiliza una instancia del problema para visualizar el comportamiento de la red, entre las secciones de prueba.

La primera sección de pruebas plantea un esquema mono objetivo donde WS define todo el peso a un objetivo a la vez, el objetivo a minimizar tendrá una preferencia de w_j =1 considerado que $\sum_{i=1}^{m} w_i = 1$.

Además, se analizan tres casos para alguna de las variables en la tupla (k, n):

1. Se define a priori n.

2. Se define a priori k.

3. No se definen a priori n y k.

La segunda sección de pruebas considera solo el tercer caso donde no se definen valores a la tupla (k, n). La red neuronal realiza múltiples entrenamientos para resolver los subproblemas generados al variar los valores w_i, estos valores representan la importancia de los m objetivos y a diferencia de la primera sección donde el valor 1 se fija en alguno de los objetivos.

Al terminar de resolver un subproblema específico entonces se resuelve un subproblema cercano al anterior al introducir variaciones pequeñas en el vector de preferencias, el nuevo problema toma los pesos de Ptr-Net y Critic-Net del subproblema anterior como punto de partida; solo se cuida en reiniciar los valores del optimizador y el paso de aprendizaje. La misma estrategia se utiliza para resolver los demás subproblemas definidos por el vector de preferencias.

Cada subproblema resuelto presenta dos gráficas para denotar los resultados. La primera gráfica muestra los valores de los objetivos individualmente en la configuración (k, n) y el valor WS con los valores de preferencia asignados a w_{Pr} y w_r. La segunda grafica denota la ubicación de la solución encontrada en el espacio de solución con valores normalizados. Finalmente, una gráfica exhibe todas las soluciones encontradas conformando un frente de Pareto.

Los valores del optimizador ADAM fueron reiniciados en el entrenamiento de cada subproblema porque los cálculos del gradiente anterior quedan guardados en memoria.

5.3.1 Evaluación estadística

Las fórmulas utilizadas para evaluar las soluciones obtenidas experimentos fueron la media y varianza muestral. Denotadas por las siguientes ecuaciones.

Media aritmética:

$$\overline{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{36}$$

Varianza muestral:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}) \tag{37}$$

Se realizaron 30 ejecuciones para cada par de valores de pesos del vector w con una secuencia de entrada de $|N| = 10$.

5.4 Optimización multi objetivo

La optimización multi objetivo (MOP, por sus siglas en inglés) busca resolver problemas con más de 2 los objetivos en conflicto, esto es, el mejorar en un objetivo implica el detrimento de algún otro. Una definición ampliamente aceptada es propuesta por Osyczka (1985):

Encontrar un vector de variables de decisión las cuales satisfacen restricciones y optimiza un vector función del cual sus elementos representan las funciones objetivo. Estas funciones forman una descripción matemática del criterio de desempeño las cuales están en conflicto una con otras. Por lo tanto, el término “optimizar” significa encontrar una solución la cual podría dar valores a todas las funciones objetivo que son aceptables para el diseñador.

Una solución es un vector n-dimencional en algún universo Q denominado espacio de decisión,

$$X = [x_1, x_2, \dots x_n]^T. \tag{38}$$

Sea un vector o vectores (soluciones) con q desigualdades de restricción, p igualdades de restricción y un vector de función $F(X) = [f_1(X), f_2(X), \dots, f_k(X)]$ donde $f_i: \mathbb{R}^k \longrightarrow \mathbb{R}$. Entonces, se busca el conjunto de soluciones que satisfaga las restricciones y además, encuentren el valor óptimo para el vector F.

En los problemas multi objetivos, el conjunto de soluciones pueden ser incontable (dominios continuos), sí es el caso, el usuario determinara la solución que le sea más útil. En ocasiones, los MOP pueden tener las mismas (conmensurable) o distintas (no conmensurable) unidades entre las funciones objetivo. En cualquier caso, el problema general es NP-completo (Bäck, 1996).

5.4.1 Terminología de Pareto

Al tener diversos objetivos, la definición de óptimo cambia porque se busca un conjunto que refleje los distintos niveles de compromiso entre las soluciones. Para esto es usado el conjunto de Pareto.

Una solución x es Pareto óptimo con respecto al universo Q de soluciones, si y solo si, no existe $x' \in Q$ para el cual $o = F(x) = \big(f_1(x), \dots, f_k(x)\big)$ domina $u = F(x) = \big(f_1(x'), \dots, f_k(x')\big)$.

Un vector solución $u = (u_1, \dots, u_k)$ domina a otro vector $o = (o_1, \dots, o_k)$ denotado por $u \leq o$, si y solo si, u es parcialmente menor que o, es decir, $\forall i \in \{1, \dots, k\}, u_i \leq o_i \ \wedge \ \neg \exists \ i \in \{1, \dots, k\} | u_i < o_i$.

Pareto óptimo: es el conjunto $\mathcal{P}^*$ de todas las soluciones no dominadas, tal que:

$$\mathcal{P}^* \ \coloneqq \{x \in \mathcal{F} | \ \exists \ x' \in Q \quad F(x') \ \leq \ F(x)\} \tag{39}$$

El frente de Pareto $\mathcal{PF}$ es la representación gráfica del conjunto Pareto óptimo, tal que:

$$\mathcal{PF}^* \coloneqq \{ \ u = F(x) \ | x \ \in \mathcal{P}^* \} \, . \tag{40}$$

Además, las soluciones de Pareto no dominadas se pueden dividir en débiles y estrictas. Para la primera definición tenemos que una solución $x^* \in \vartheta$ es débil Pareto óptimo sí,

$$\nexists\, x \in Q\ \forall i \in \{1, \dots, k\} |\ f_i(x) < f_i(x^*) \tag{41}$$

Y para la segunda tenemos que una solución $x^* \in Q$ es estrictamente Pareto óptimo sí,

$$\nexists\, x \in Q\ \forall i \in \{1, \dots, k\} |\ f_i(x) \leq f_i(x^*) \tag{42}$$

En MOP el mínimo global son todas las soluciones que pertenecen al conjunto de Pareto óptimo $\mathcal{P}^*$, aunque no existe una definición universalmente (Coello Coello et al., 2007).

5.4.2 Vector ideal, vector utópico y Convexidad

Vector ideal: este vector es la solución ideal (utópica), se consigue optimizando de manera individual cada objetivo por separado y respetando todas las restricciones existentes, que garantizan una solución válida. El vector ideal es $f^{id} = \left[f_1^{id}, f_2^{id}, \dots, f_k^{id}\right]^T$ un vector $\mathbb{R}^k$ en el espacio solución donde f_i^{id} es la solución óptima para la función i.

Vector utópico: a pesar que el vector ideal es considerado un vector utópico, autores como Burke y Graham (2014) consideran que el vector utópico no puede ser alcanzado, tal que, el vector utópico es $f^{uto} = f_i^{id} - \varepsilon\ \forall\ i \in \{1, \dots, \mathrm{k}\}, \varepsilon > 0$.

Convexidad: Una función ρ sobre el dominio $\mathbb{R}$ es convexa sí para cualquiera dos vectores $x_1, x_2 \in \mathbb{R}$,

$$\rho(\vartheta x_1 + (1 - \vartheta)x_2) \leq \vartheta\rho(x_1) + (1 - \vartheta)\rho(x_2) \tag{43}$$

donde ϑ es un escalar en el rango de $0 \leq \vartheta \leq 1$.

Se puede observar en la Figura 16 una representación 2D de las posiciones del frente de Pareto convexo y los vectores ideal y utópico.

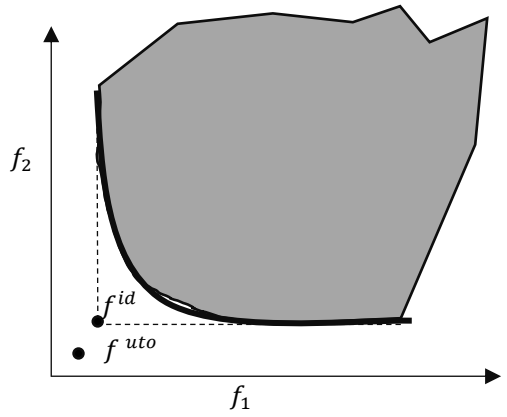


Figura 16. Posiciones de los vector ideal, utópico y frente convexo.

5.4.3 Clasificación de los algoritmos para la combinación combinatoria

El área de investigación operativa clasifica los tipos de algoritmos que resuelven MOP. Cohon y Marks (1975) establece la las siguiente clasificación:

- Articulación de preferencia a priori: las decisiones se realizan antes de buscar y engloba a todos los algoritmos que establecen una meta o un ordenamiento por preferencia a priori de las funciones.

- Articulación de preferencias a posteriori: las decisiones se realizan después de buscar y no requieren información adicional por parte del usuario tomador de decisiones. Entre sus técnicas se encuentras los métodos clásicos de optimización multi objetivo y suma ponderada.

- Articulación de preferencias progresiva: las decisiones se realizan mientras sucede la búsqueda, el usuario interviene durante la búsqueda.

5.4.4 Comparativa con otros algoritmos

El desempeño de la red neuronal se compara con algoritmos ampliamente utilizados para afrontar problemas multi objetivo, la idea es entender su comportamiento y la capacidad de resolver problemas de este tipo.

5.4.4.1 Algoritmos genéticos

García Hernández (2019) utilizó algoritmos genéticos para resolver el problema de optimización multi objetivo en AR-RRNS y determinó que MOCell y NSGA-II fueron los algoritmos con mejores desempeño. La configuración de los parámetros es:

- Evaluación máxima: 25,000,
- Probabilidad de cruzamiento $p_c = 0.9$,
- Probabilidad del operador mutación $p_N = 1/N$,
- Cruzamiento de un punto,
- Mutación de volteo de un bit,
- Selección de torneo binario.

Adicionalmente, también se compara con un algoritmo genético simple con una función de agregación similar a la utilizada en la Ptr-Net, con los mismos hiper parámetros.

La codificación de una solución consiste de dos cromosomas, el primero cromosoma representa las nubes a utilizar y el segundo define la codificación binaria de k, véase Figura 17.

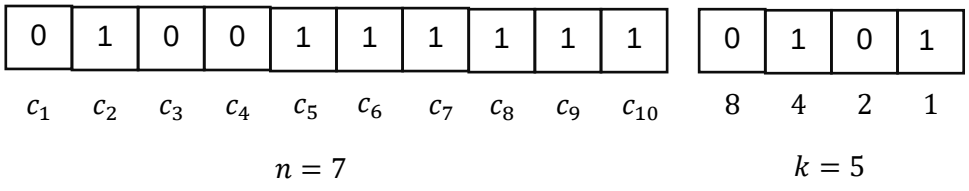


Figura 17. Codificación de la solución para GA.

5.4.4.2 Ramification y poda (Branch and bound)

Ramificacion y poda es una técnica para explorar un grafo dirigido similar a la técnica de volver hacia atrás (*Backtraking)*, la técnica consiste en analizar las ramas con soluciones más prometedoras y podar el resto. En cada nodo del árbol se calcula una cota, si la cota del nodo es peor que la mejor solución encontrada hasta ese momento entonces se descarta el nodo, no es necesario explorar esa parte del árbol (Brassard y Bratley, 1996). Existen diversas maneras de explorar el árbol búsqueda: anchura, profundidad, montículo binario, heurística, etc.

En este caso, la implementación hace uso de un montículo binario mínimo para alojar los nodos por el valor de su cota, el nodo inicial es la cantidad mínima de n y k. La cota está conformada con el mínimo valor alcanzable de los objetivos individuales.

5.4.4.3 Heurística

Se ordenan los CSP's por el valor de sus parámetros y en ese orden son considerados y evaluados, se almacena la mejor solución hasta el momento, en caso de empeorar la solución se detiene la búsqueda y se incrementa n. La búsqueda continua hasta que $k = n = |N|$.

Capítulo 6 Resultados y análisis

En este capítulo se presentan los resultados obtenidos con la red Ptr-Net y otras estrategias. Inicialmente, Ptr-Net soluciona un problema mono objetivo donde se trabaja con una sola función objetivo a la vez. Posteriormente, Ptr-Net encuentra soluciones multi objetivo donde se optimizan ambas funciones a la vez. Finalmente, se compara el desempeño de Ptr-Net con otros algoritmos. Una importante aclaración sobre los valores n_s y k_s es que se obtienen al decodificar la salida de la Ptr-Net, mientras que n y k son parámetros del sistema AR-RRNS; por lo tanto $n_s = n$ y $k_s = k$.

6.1 Optimizando una variable

El problema de optimización mono objetivo considera tres casos:

- El número de nubes es definido, $n = 10$, y la Ptr-Net determina el valor k de réplicas,
- El número de réplicas es definido, $k = 2$, y la Ptr-Net decide el número n de nubes,
- Ningún valor de k y n es asignado, y la Ptr-Net elige ambos valores.

Los valores de k y n se obtienen a través de la secuencia $\pi^{(v)}$. Los resultados son presentados en cuatro gráficas para cada solución, cada figura muestra la solución encontrada por el modelo para comparar las diferencias entre las distintas funciones objetivo.

a) Solución encontrada respecto a los valores de Pr,

b) Solución encontrada respecto a los valores de R,

c) Funciones objetivos normalizadas y la línea verde representa a WS,

d) Intercambio entre R_n y Pr_{n_ln} si se considera WS con $w_{Pr} = 0.5$ y $w_R = 0.5$.

6.1.1 Probabilidad de perdida de la información $Pr(k,n)$

En esta subsección, los valores del vector de preferencias son $w_{Pr} = 1$ y $w_R = 0$, dando preferencia absoluta a Pr_{n_ln}. Además, los valores $R_{max} = 10$, $R_{min} = 2$, $R = 10.0$, $WS_{max} = 1.0$, $WS_{min} = 0.0$ y $WS = 0.0$ son los mismos para todos los subcasos de esta sección.

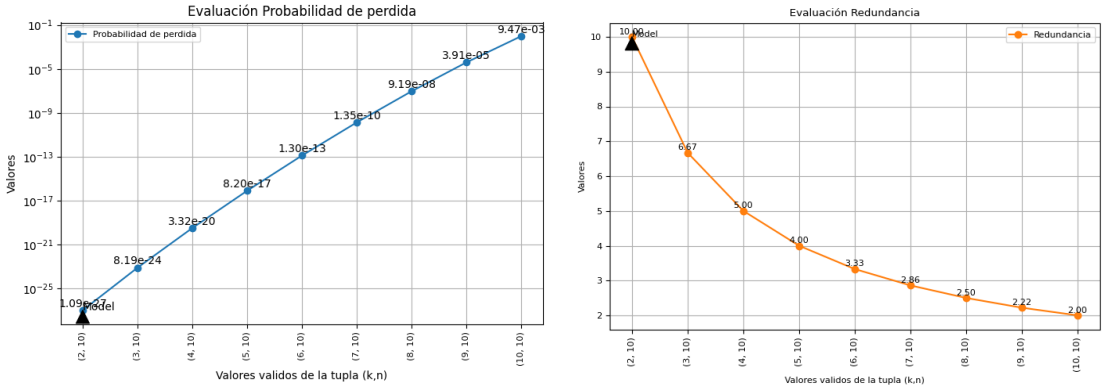
6.1.1.1 $n = 10, w_{Pr} = 1$ y $w_R = 0$

La salida inferida por el modelo es (2, 10) con $k = 2$, donde:

$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -62.08$.

$Pr_{n_ln} = \frac{-62.08+62.08}{-4.66+62.08} = 0.00$, $R_n = \frac{10.0-2.0}{10.0-2.0} = 1.00$ y $WS_n = \frac{0.00-0.00}{1.00-0.00} = 0.00$.

$Deg(P_r) = \frac{1.0924^{e-27}}{1.0924^{e-27}} = 1.00$, $Deg(Pr_{ln}) = \frac{|-6.21e^{+01}|}{|-6.21e^{+01}|} = 1.00$ y $Deg(R) = \frac{10.00}{2.00} = 5.00$.



a) b)

c) d)

Figura 18. WS = $Pr_{n\_ln} * 1 + R_n * 0$, (d) Verde WS = $Pr_{n\_ln} * 0.5 + R_n * 0.5$, n = 10.

Esta instancia del problema establece una restricción de $n = 10$, la Ptr-Net genera una secuencia $\pi^{(v)}$ con los valores de $k_s = 2$ y $n_s = 10$, esta solución ofrece la mínima probabilidad de perdida de información Pr y satisface la restricción.

La

Figura **18**.a muestra que la solución encontrada por la Ptr-Net es óptima. En la

Figura **18**.b, la redundancia ha sido completamente ignorada tomando el peor valor que podría obtener. En la

Figura **18**.c el objetivo Pr_{n_ln} es igual al objetivo WS. La

Figura **18**.d considera un vector w con distinto conjunto de preferencias para la función WS el mínimo se encontraría en (4,10).

6.1.1.2 $\mathbf{k = 2}$,$\mathbf{w_{Pr} = 1}$ y $\mathbf{w_R = 0}$

La solución inferida por el modelo es (2, 10) con $n = 10$, donde:

$Pr_{ln}max = -5.649$, $Pr_{ln}min = -62.08$ y $Pr_{nl} = -62.08$.

$Pr_{n\_ln} = \frac{-62.08+62.08}{-5.649+62.08} = 0.00$, $R_n = \frac{10.0-2.0}{10.0-2.0} = 1.00$ y $WS_n = \frac{0.00-0.00}{1.00-0.00} = 0.00$.

$Deg(P_r) = \frac{1.092e^{-27}}{1.092e^{-27}} = 1.00$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-62.08|} = 1.00$ y $Deg(R) = \frac{10.00}{2.00} = 5.00$.

Esta instancia del problema establece una restricción de $k = 2$, la Ptr-Net genera una secuencia $\pi^{(v)}$ con los valores de $k_s = 2$ y $n_s = 10$, P_r es óptima porque $Pr_{ln} = Pr_{ln}min$ y satisface la restricción.

En la Figura 19.a se muestra que la solución encontrada por la Ptr-Net es óptima. En la Figura 19.b, la redundancia ha sido completamente ignorada tomando el peor valor que podría obtener. En la Figura 19.c, $WS = Pr_{n\_ln}$ el objetivo ponderado. En la Figura 19.d sí se considera un vector w con distinto conjunto de preferencias para la función WS el mínimo se encontraría en otra solución, y la actual sería la peor.

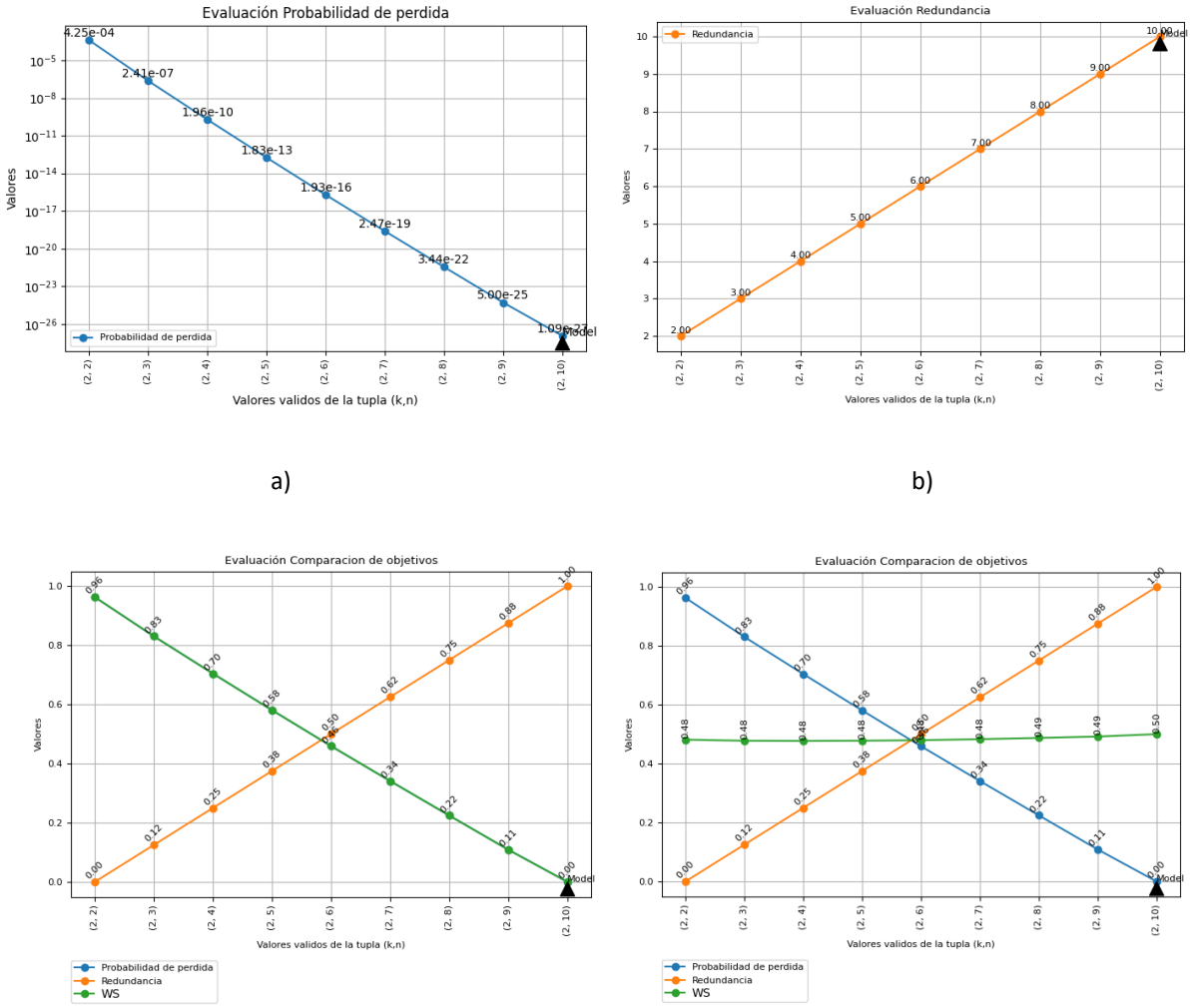


Figura 19. WS $= Pr_{n\_ln} * 1 + R_n * 0$, Verde WS $= Pr_{n\_ln} * 0.5 + R_n * 0.5$, k = 2.

6.1.1.3 $\mathbf{w_{Pr} = 1}$ y $\mathbf{w_R = 0}$

La salida inferida por el modelo es (2, 10) con $n = 10$ y $k = 2$, donde:

$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{nl} = -62.08$.

$Pr_{n\_ln} = \frac{-62.08+62.08}{-4.66+62.08} = 0.00$, $R_n = \frac{10.0-2.0}{10.0-2.0} = 1.00$ y $WS_n = \frac{0.00-0.00}{1.00-0.00} = 0.00$.

$Deg(P_r) = \frac{1.0924^{e-27}}{1.0924^{e-27}} = 1.00$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-62.08|} = 1.00$ y $Deg(R) = \frac{10.00}{2.00} = 5.00$.

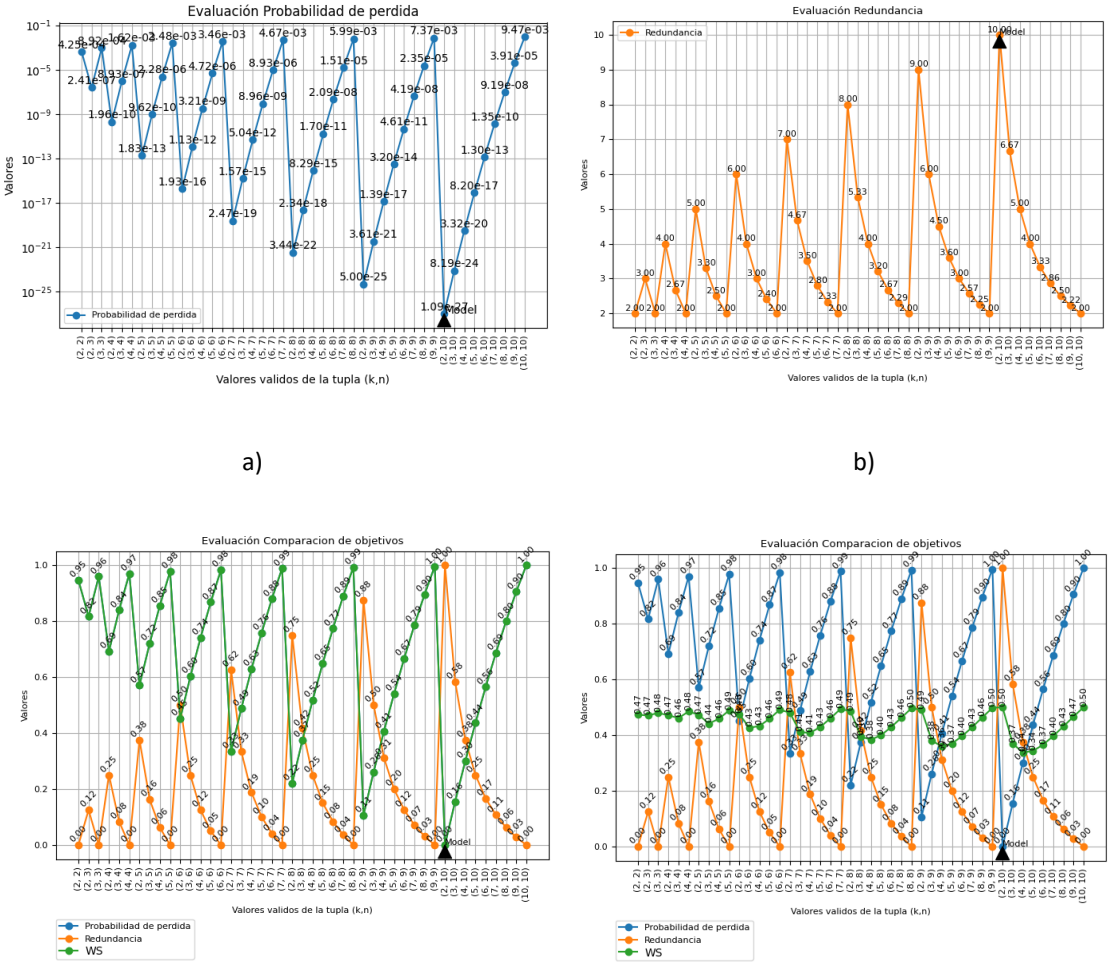


Figura 20. WS $= Pr_{n\_ln} * 1 + R_n * 0$, Verde WS $= Pr_{n\_ln} * 0.5 + R_n * 0.5$.

Esta instancia del problema no establece restricciones de igualdad, la Ptr-Net genera una secuencia $\pi^{(v)}$ con los valores $k_s = 2$, $n_s = 10$, con $Pr_{ln} = Pr_{ln}min$.

La Figura 20.a muestra que la solución encontrada por la Ptr-Net es óptima. En la Figura 20.b, la redundancia R ha sido completamente ignorada tomando el peor valor que podría obtener. En la Figura 20.c el objetivo ponderado $Pr_{n\_ln} = WS$ es una función no convexa a pesar de ello se logra encontrar la solución óptima. En la Figura 20.d, un vector w con distinto conjunto de preferencias representa el intercambio entre objetivos para la función WS y la solución óptima se encontraría en (4,10).

6.1.2 Redundancia R

Esta sección considera un vector de preferencias con valores fijo donde $w_{Pr} = 0$ y $w_R = 1$, es decir, R cuenta con preferencia absoluta. Además, los valores $R_{max} = 10$, $R_{min} = 2$, $R = 2.0$, $WS_{max} = 1.00$, $WS_{min} = 0.0$ y $WS = 0.00$ son los mismos para todos los subcasos.

6.1.2.1 $\mathbf{n = 10, w_{Pr} = 0}$ y $\mathbf{w_R = 1}$

La salida inferida por el modelo es: (10, 10) con k = 10, donde:

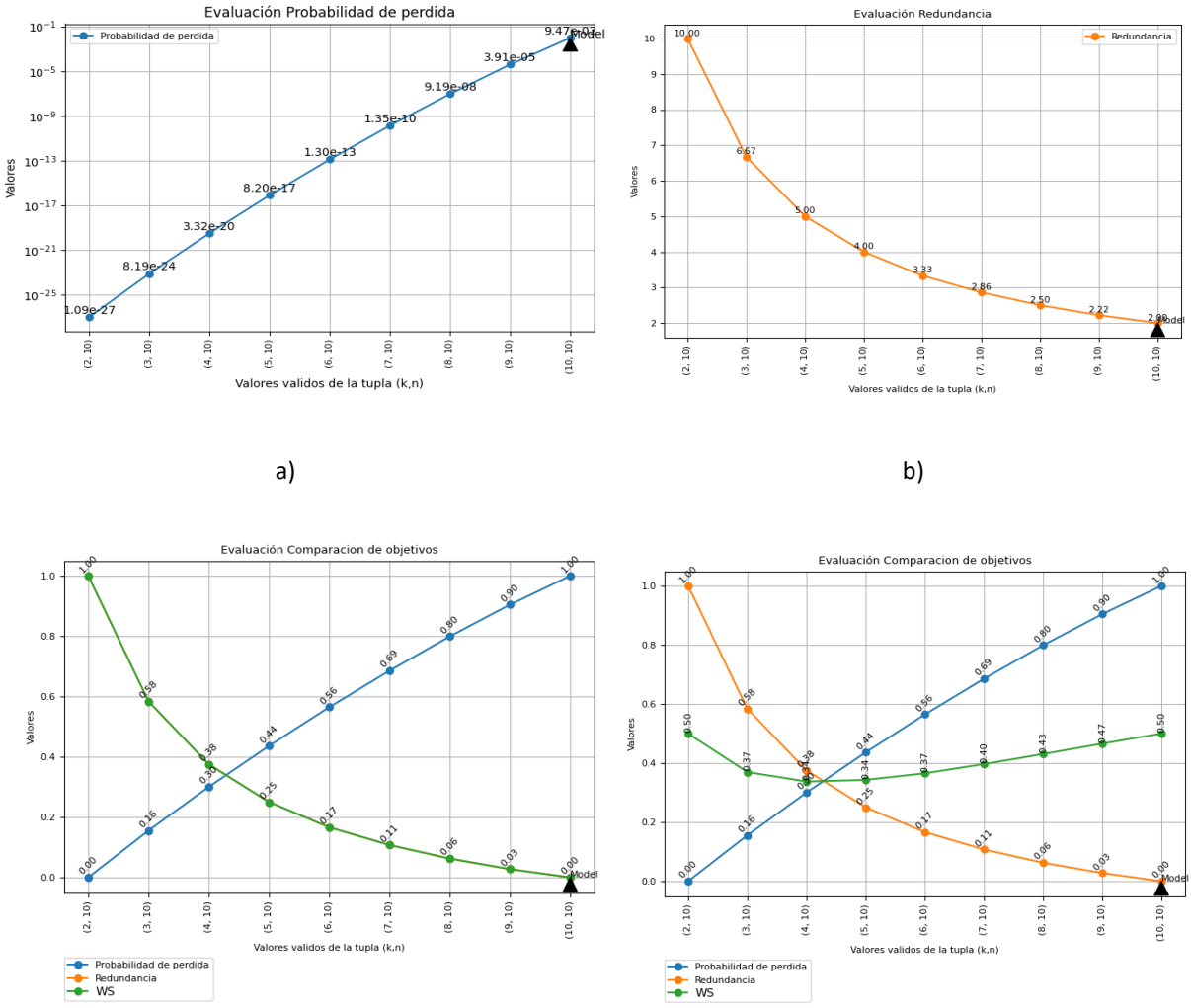
$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{nl} = -4.66$.

$Pr_{n\_ln} = \frac{-4.660+62.08}{-4.660+62.08} = 1.00$, $R_n = \frac{2-2}{10-2} = 0.00$ y $WS_n = \frac{0.00-0.00}{1.00-0.00} = 0.00$

$Deg(P_r) = \frac{9.466^{e-03}}{1.092e^{-27}} = 8.666e^{+24}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-4.66|} = 13.32$ y $Deg(R) = \frac{2.00}{2.00} = 1.00$

Esta instancia del problema establece una restricción de $n = 10$, la Ptr-Net generar una secuencia $\pi^{(v)}$ con los valores de $k_s = 10$ y $n_s = 10$, esta solución ofrece la mínima redundancia $R = R_{min}$ y satisface la restricción.

En la Figura 21.a la solución encontrada por la Ptr-Net es la peor para el objetivo Pr pero esta es óptima con respecto a $R = Rmin$, véase Figura 21.b. En la Figura 21.c, el objetivo R_n es igual a WS. En la Figura 21.d sí se considera un vector w con distinto conjunto de preferencias para la función WS el valor óptimo se encontraría en otra solución (4,10).



a) b)

c) . d)

Figura 21. WS = $Pr_{n\_ln} * 0 + R_n * 1$, verde WS = $Pr_{n\_ln} * 0.5 + R_n * 0.5$, n = 10.

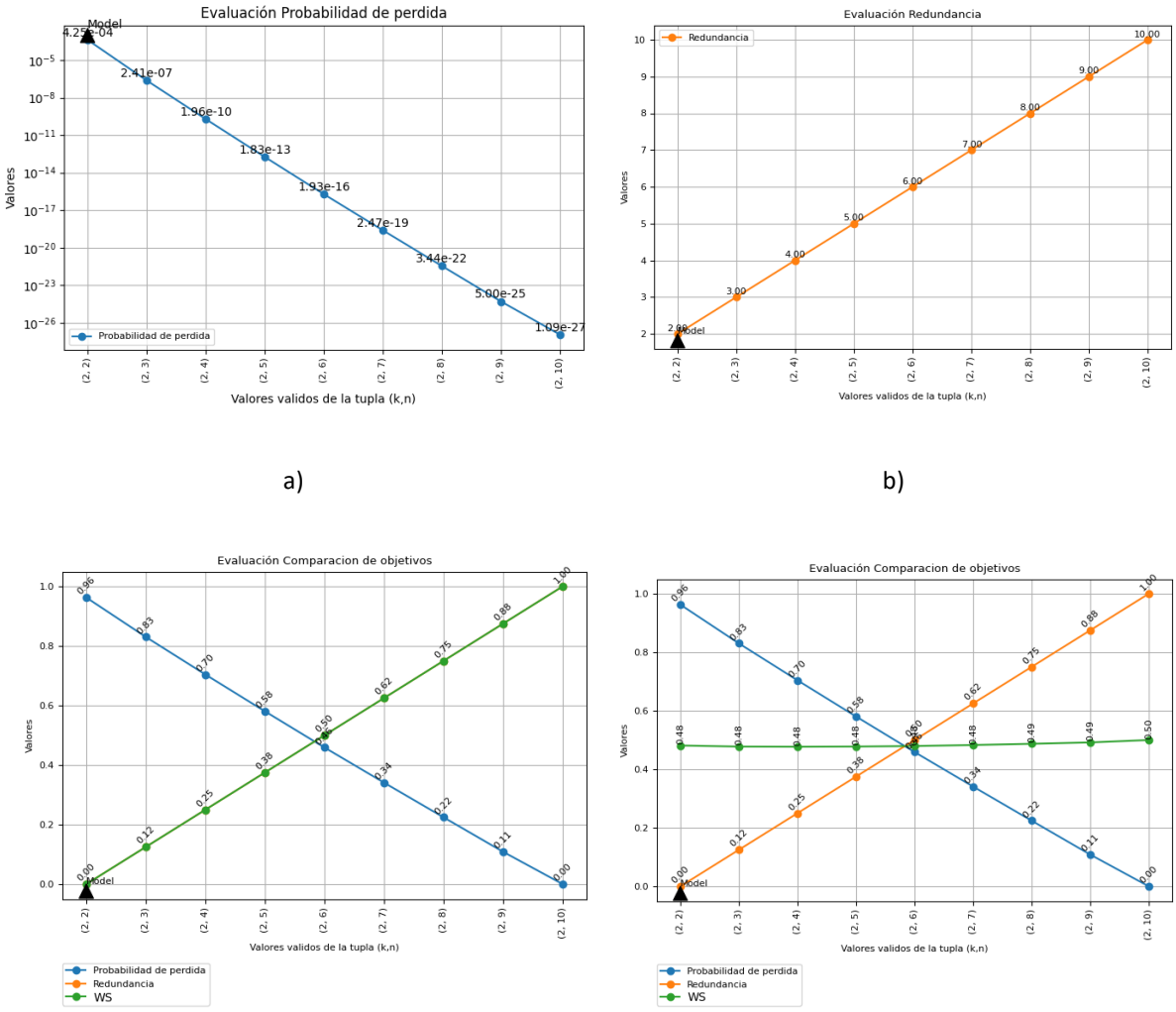
6.1.2.2 $\mathbf{k = 2, w_{Pr} = 0}$ y $\mathbf{w_R = 1}$

La solución inferida por el modelo es (2,2) con $n = 2$, donde:

$Pr_{ln}max = -5.65$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -5.65$.

$Pr_{n\_ln} = \frac{-5.649+62.08}{-5.649+62.08} = 1.00$, $R_n = \frac{2.0-2.0}{10.0-2.0} = 0.00$ y $WS_n = \frac{0.00-0.00}{1.00-0.00} = 0.00$.

$Deg(P_r) = \frac{3.520e^{-03}}{1.092e^{-27}} = 3.223e^{+24}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-5.649|} = 10.98$ y $Deg(R) = \frac{2.00}{2.00} = 1.00$.



a) b)

c) d)

Figura 22. WS $= Pr_{n\_ln} * 0 + R_n * 1$, verde WS $= Pr_{n\_ln} * 0 + R_n * 1$, $k = 2$.

Esta instancia del problema establece una restricción de $k = 2$, la Ptr-Net genera una secuencia $\pi^{(v)}$ con los valores de $k_s = 2$ y $n_s = 2$, esta solución es óptima porque $R = Rmin$ y satisface la restricción.

La Figura 22.a muestra que la solución encontrada es la peor para Pr pero es óptima $R = Rmin$, véase Figura 22.b. En la Figura 22.c el objetivo $R_n = WS$ y en la Figura 22.d sí se considera un vector w con distinto conjunto de preferencias para la función WS el mínimo se encontraría en otra solución, siendo la actual la peor.

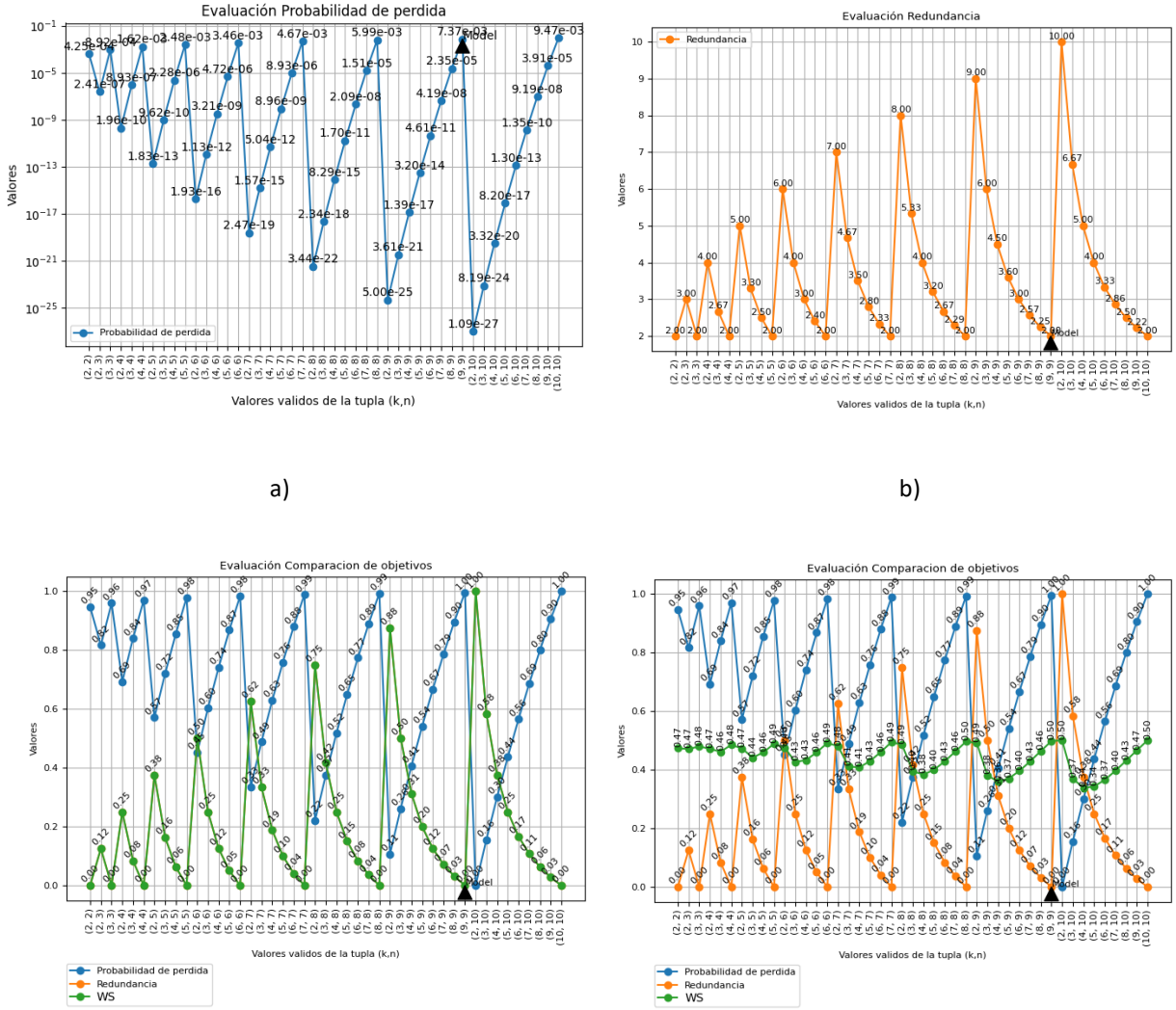
6.1.2.3 $\mathbf{w_{Pr} = 0}$ y $\mathbf{w_R = 1}$

La salida inferida por el modelo es (9, 9) con $k = 9$ y $n = 9$, donde:

$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -4.91$.

$Pr_{n\_ln} = \frac{-4.911+62.08}{-4.660+62.08} = 0.996$, $R_n = \frac{2.0-2.0}{10.0-2.0} = 0.00$ y $WS_n = \frac{0.00-0.00}{1.00-0.00} = 0.00$.

$Deg(P_r) = \frac{7.367e^{-03}}{1.092e^{-27}} = 6.744e^{+24}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-4.911|} = 12.64$ y $Deg(R) = \frac{2.00}{2.00} = 1.00$



a) b)

c) d)

Figura 23. WS $= Pr_{n\_ln} * 0 + R_n * 1$, verde WS $= Pr_{n\_ln} * 0 + R_n * 1$.

Esta instancia del problema no establece alguna restricción de igualdad, la Ptr-Net genera una secuencia $\pi^{(v)}$ con los valores de $k_s = 9$, $n_s = 9$ y $R = Rmin$.

En Figura 23.a la solución encontrada es mala pero no la peor para Pr. En Figura 23.b es óptima con respecto a $R = Rmin$. En Figura 23.c tenemos que $R_n = WS$, pero algo interesante sucede es que R_n cuenta con múltiples mínimos cuando $k = n$, y al no depender de Pr_{n_ln} puede tomar cualquiera de estas soluciones y alcanzar el mínimo. En Figura 23.d sí se considera un vector w con distinto conjunto de preferencias para la función WS representa este intercambio y la solución se encontraría en (4,10).

6.1.3 Resolviendo subproblemas derivados de WO para multiobjetivo

La metodología consiste en resolver múltiples problemas secuencialmente. Después de encontrar una solución para el j-ésimo problema, los pesos del vector w son modificados para obtener valores cercanos. Finalmente, los parámetros de la Ptr-Net del problema $j-1$ se usan para encontrar una solución al problema j.

6.1.3.1 $\mathbf{w_{Pr}} = \mathbf{1.0}$ y $\mathbf{w_r} = \mathbf{0.0}$

La salida inferida por el modelo es (2,10) con $k = 2$ y $n = 10$, donde:

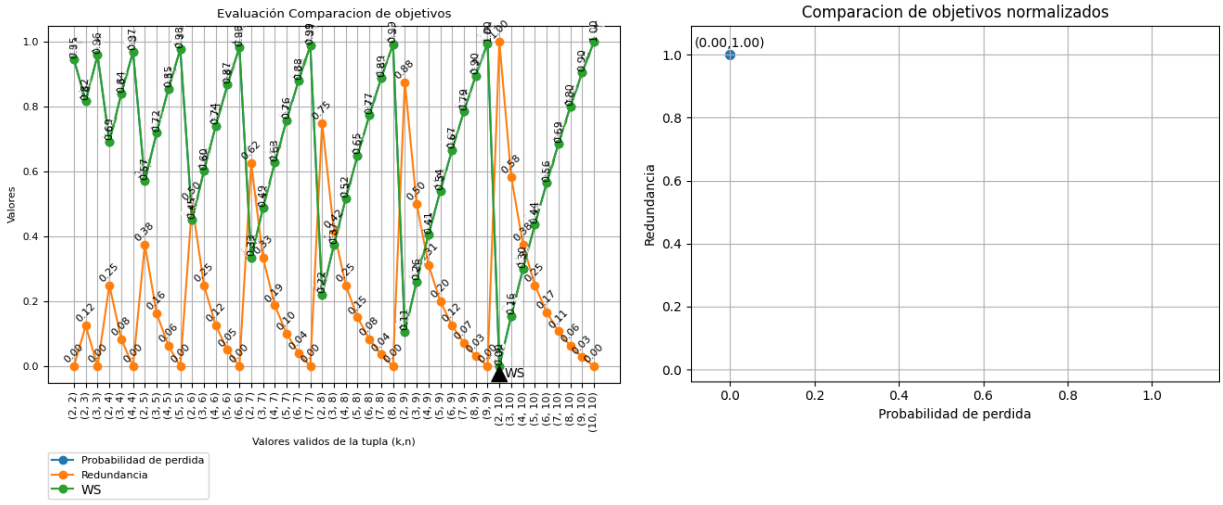
$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{nl} = -62.08$.

$R_{max} = 10.0$, $R_{min} = 2.0$ y $R = 10.0$.

$WS_{max} = 1.00$, $WS_{min} = 0.00$ y $WS = 0.00$.

$Pr_{n_{l}n} = \frac{-62.08+62.08}{-4.66+62.08} = 0.00$, $R_n = \frac{10.0-2.0}{10.0-2.0} = 1.00$ y $WS_n = \frac{0.00-0.00}{1.00-0.00} = 0.00$.

$Deg(Pr) = \frac{1.0924^{e-27}}{1.0924^{e-27}} = 1.00$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-62.08|} = 01.00$ y $Deg(R) = \frac{10.00}{2.00} = 5.00$.



a) b)

Figura 24. $\mathrm{WS} = Pr_{n\_ln} * 1.0 + R_n * 0.0$.

La Figura 24.a muestra que el modelo obtiene le valor óptimo $WS_n = Pr_{n\_ln} = 0$. Figura 24.b presenta su representacion en el espacio solucion con las fuciones Pr_{n_ln} y R_n.

6.1.3.2 $\mathbf{w_{Pr} = 0.9}$ y $\mathbf{w_r = 0.1}$

La salida inferida por el modelo es (2, 10) con $k = 2$ y $n = 10$, donde

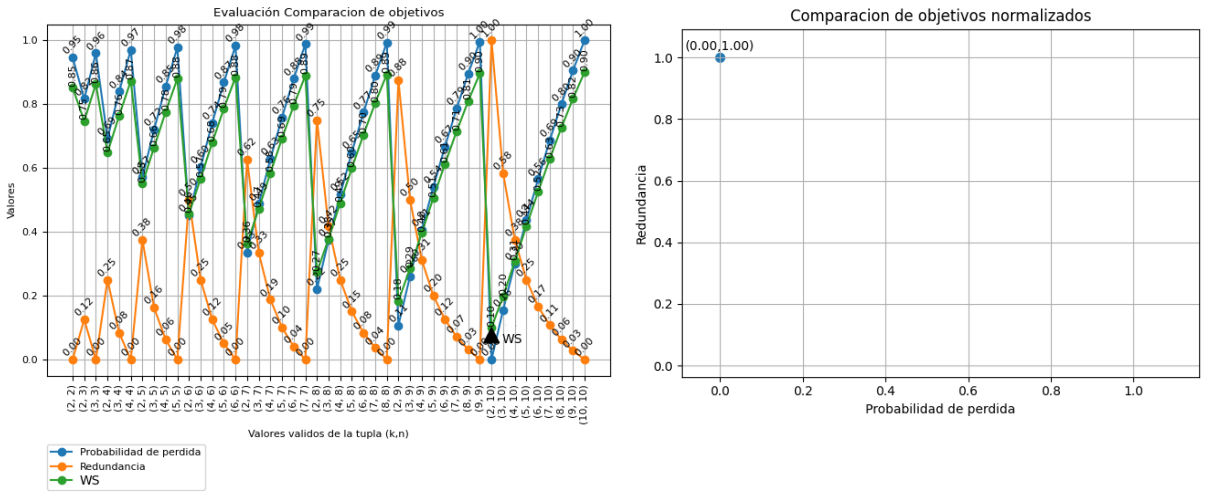
$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -62.08$.

$R_{max} = 10.0$, $R_{min} = 2.0$ y $R = 10.0$.

$WS_{max} = 0.90$, $WS_{min} = 0.90$ y $WS = 0.10$.

$Pr_{n\_ln} = \frac{-62.08+62.08}{-4.66+62.08} = 0.00$, $R_n = \frac{10.0-2.0}{10.0-2.0} = 1.00$ y $WS_n = \frac{0.10-0.10}{0.90-0.10} = 0.00$.

$Deg(Pr) = \frac{1.0924^{e-27}}{1.0924^{e-27}} = 1.00$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-62.08|} = 01.00$ y $Deg(R) = \frac{10.00}{02.00} = 5.00$.



a) b)

Figura 25. $WS = Pr_{n\_ln} * 0.9 + R_n * 0.1$.

La Figura 25.a muestra que se obtiene el valor óptimo $WS_n = 0$ y la Figura 25.b presenta su representación en el espacio solución con las funciones Pr_{n_ln} y R_n.

6.1.3.3 $\mathbf{w_{Pr} = 0.8}$ y $\mathbf{w_r = 0.2}$

La salida inferida por el modelo es (2, 10) con $k = 2$ y $n = 10$, donde:

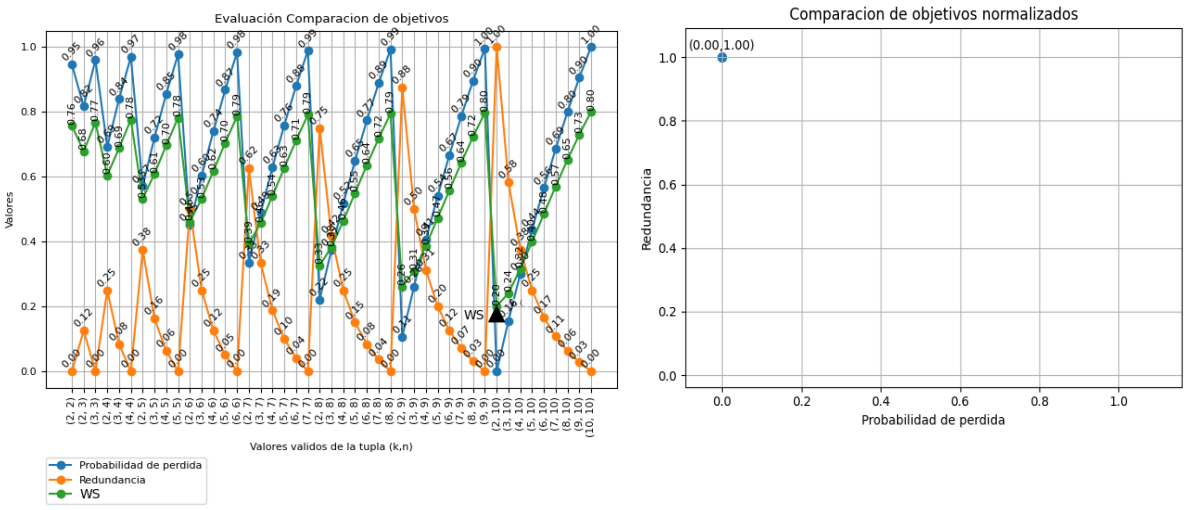
$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -62.08$.

$R_{max} = 10$, $R_{min} = 2$ y $R = 10.0$.

$WS_{max} = 0.80$, $R_{min} = 0.20$ y $WS = 0.20$.

$Pr_{n\_ln} = \frac{-62.08+62.08}{-4.66+62.08} = 0.00$, $R_n = \frac{10.0-2.0}{10.0-2.0} = 1.00$, $WS_n = \frac{0.20-0.20}{0.80-0.20} = 0.00$

$Deg(P_r) = \frac{1.0924^{e-27}}{1.0924^{e-27}} = 1.00$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-62.08|} = 01.00$, $Deg(R) = \frac{10.00}{2.00} = 5.00$



a) b)

Figura 26. WS $= Pr_{n\_ln} * 0.8 + R_n * 0.2$.

La Figura 26.a muestra que se obtiene le valor óptimo $WS_n = 0$ y la Figura 26.b presenta su representación en el espacio solución con las funciones $\boldsymbol{Pr_{n\_ln}}$ y $\boldsymbol{R_n}$.

6.1.3.4 $\mathbf{w_{Pr} = 0.7}$ y $\mathbf{w_r = 0.3}$

La salida inferida por el modelo es (3, 10) con $k = 3$ y $n = 10$, donde:

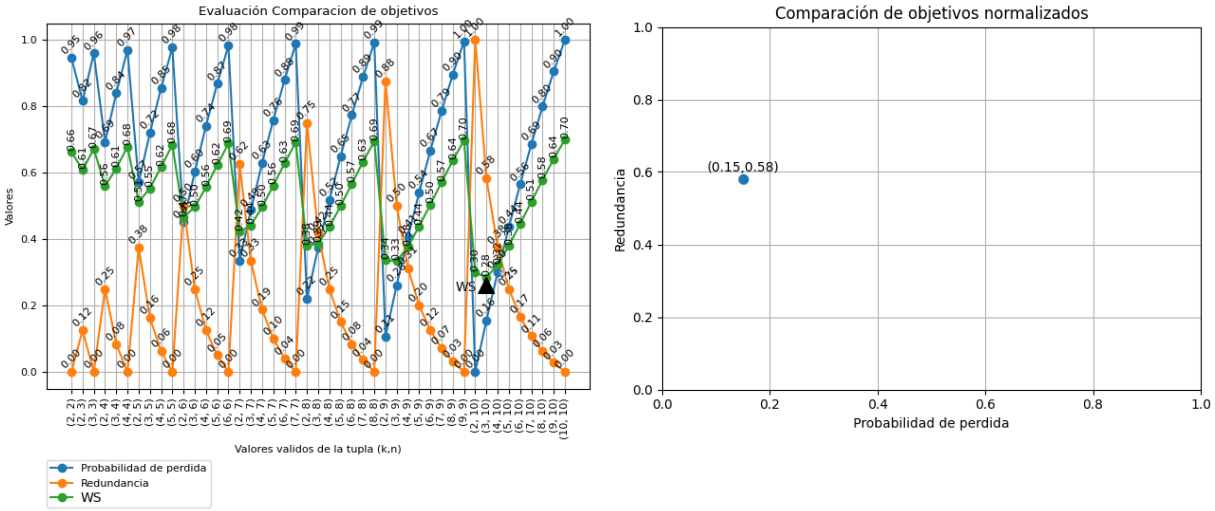
$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -53.2$.

$R_{max} = 10$, $R_{min} = 2$ y $R = 6.67$.

$WS_{max} = 0.70$, $WS_{min} = 0.28$ y $WS = 0.284$.

$Pr_{n\_ln} = \frac{-53.2+62.08}{-4.66+62.08} = 0.155$, $R_n = \frac{6.7-2.0}{10.0-2.0} = 0.584$ y $WS_n = \frac{0.28-0.28}{0.70-0.28} = 0.00$.

$Deg(Pr) = \frac{8.19e^{-24}}{1.0924^{e-27}} = 7.49^{e+03}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-53.2|} = 1.16$ y $Deg(R) = \frac{10.00}{2.00} = 5.00$.



a) b)

Figura 27. WS $= Pr_{n\_ln} * 0.7 + R_n * 0.3$.

La Figura 27.a muestra que se obtiene el valor óptimo $WS_n = 0$ y la Figura 27.b presenta su representación en el espacio solución con las funciones $\boldsymbol{Pr_{n\_ln}}$ y $\boldsymbol{R_n}$.

6.1.3.5 $\mathbf{w_{Pr} = 0.6}$ y $\mathbf{w_r = 0.4}$

La salida inferida por el modelo es (3, 10) para $k = 3$ y $n = 10$, donde:

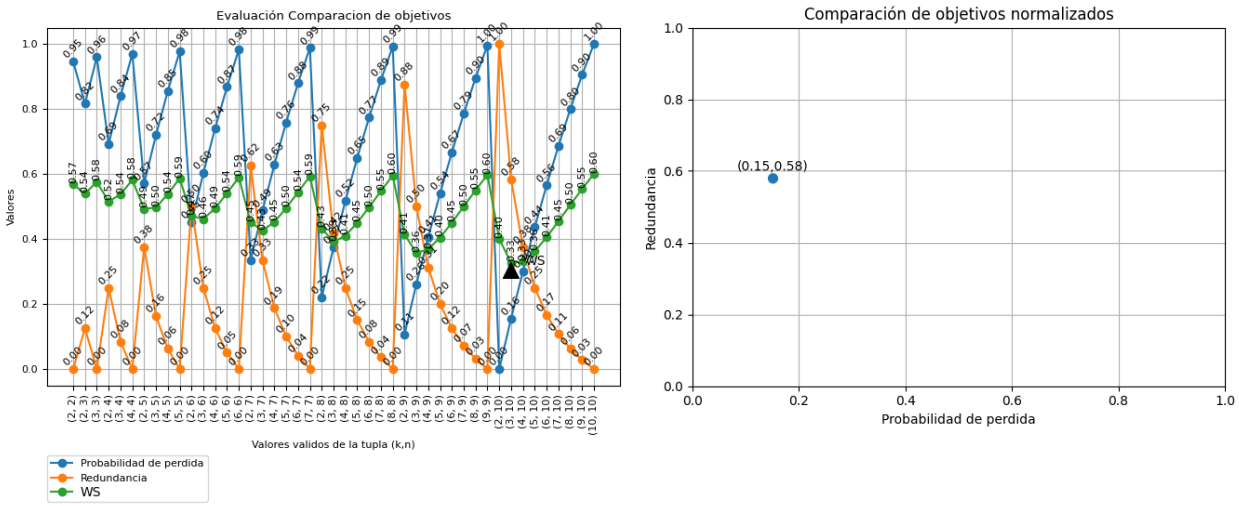
$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -53.2$.

$R_{max} = 10$, $R_{min} = 2$ y $R = 5.00$.

$WS_{max} = 0.60$, $WS_{min} = 327$ y $WS = 0.327$.

$Pr_{n\_ln} = \frac{-53.2 + 62.08}{-4.66+62.08} = 0.155$, $R_n = \frac{6.7-2.0}{10.0-2.0} = 0.584$ y $WS_n = \frac{0.327-0.327}{0.60-0.327} = 0.00$.

$Deg(P_r) = \frac{8.185e^{-24}}{1.092e^{-27}} = 7.493e^{+03}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-52.2|} = 01.16$ y $Deg(R) = \frac{6.67}{2.00} = 3.34$.



a) b)

Figura 28. WS $= Pr_{n\_ln} * 0.6 + R_n * 0.4$.

La Figura 28.a muestra que se obtiene le valor óptimo $WS_n = 0$ y la Figura 28.b presenta su representación en el espacio solución con las funciones Pr_{n_ln} y R_n.

6.1.3.6 $\mathbf{w_{Pr} = 0.5}$ y $\mathbf{w_r = 0.5}$

La salida inferida por el modelo es (4, 10) para $k = 4$ y $n = 10$, donde:

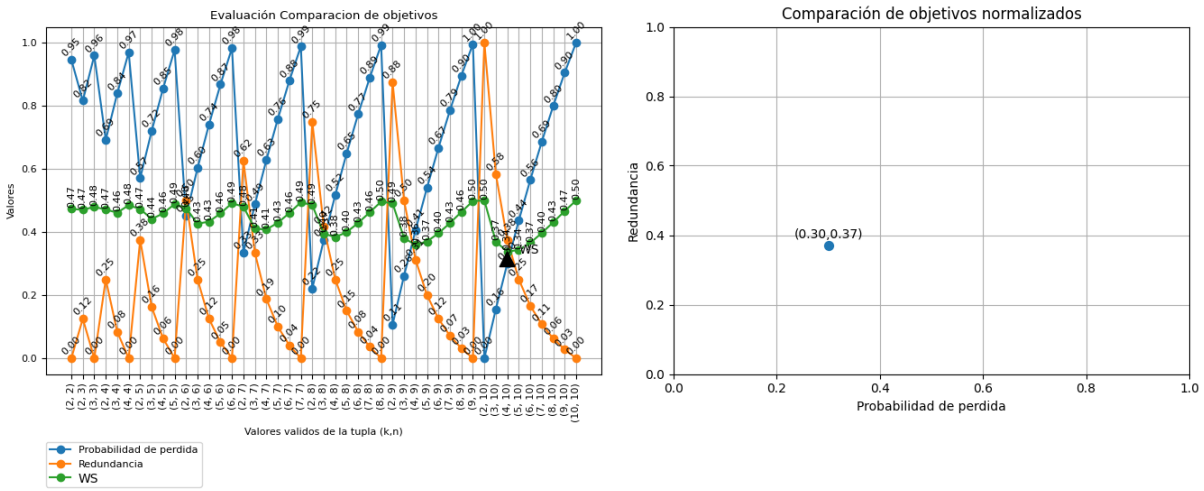
$Pr_{ln}max = -4.66$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -44.85$.

$R_{max} = 10$, $R_{min} = 2$ y $R = 5.00$.

$WS_{min} = 0.50$, $WS_{min} = 0.338$ y $WS = 0.338$

$Pr_{n\_ln} = \frac{-44.85+62.08}{-4.66+62.08} = 0.30$, $R_n = \frac{5.0-2.0}{10.0-2.0} = 0.375$ y $WS_n = \frac{0.338-0.338}{0.500-0.338} = 0.00$.

$Deg(Pr) = \frac{3.323e^{-20}}{1.092e^{-27}} = 3.042e^{+07}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-44.85|} = 1.38$ y $Deg(R) = \frac{5.00}{2.00} = 2.50$.



a) b)

Figura 29. WS $= Pr_{n\_ln} * 0.5 + R_n * 0.5$.

La Figura 29.a muestra que se obtiene el valor óptimo $WS_n = 0$ y la Figura 29.b presenta su representación en el espacio solución con las funciones Pr_{n_ln} y R_n.

6.1.3.7 $\mathbf{w_{pr} = 0.4}$ y $\mathbf{w_r = 0.6}$

La salida inferida por el modelo es (5, 10) para $k = 5$ y $n = 10$, donde:

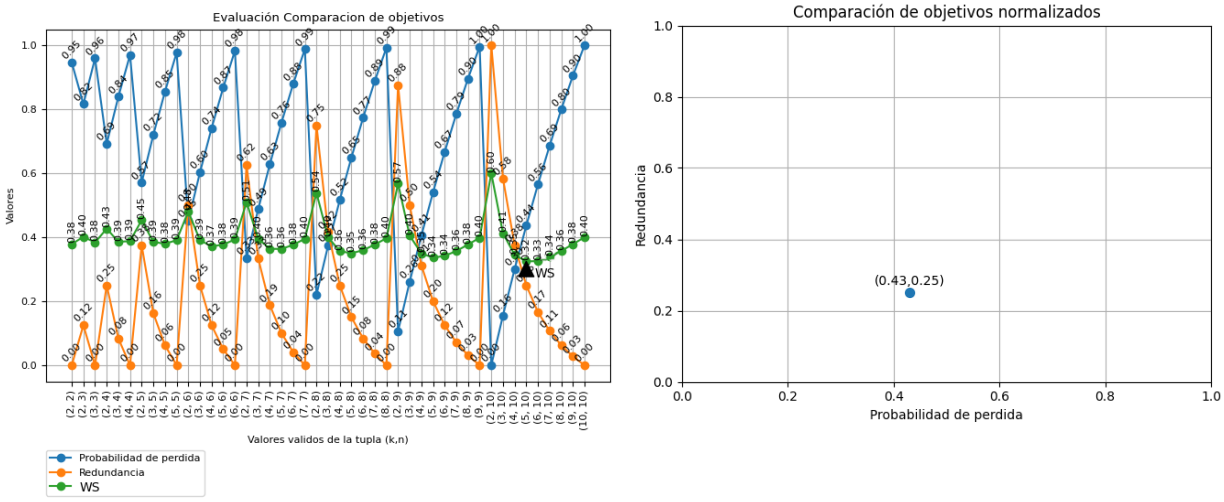
$Pr_{ln}max = -4.660$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -37.04$.

$R_{max} = 10$, $R_{min} = 2$ y $R = 4.00$.

$WS_{max} = 0.60$, $WS_{min} = 0.324$ y $WS = 0.324$.

$Pr_{n\_ln} = \frac{-37.04+62.08}{-4.66+62.08} = 0.436$, $R_n = \frac{4.0-2.0}{10.0-2.0} = 0.250$ y $WS_n = \frac{0.324-0.324}{0.600-0.324} = 0.00$.

$Deg(Pr) = \frac{8.201e^{-17}}{1.092e^{-27}} = 7.508e^{+10}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-63.08|} = 1.67$ y $Deg(R) = \frac{4.00}{2.00} = 2.00$.



a) b)

Figura 30. WS $= Pr_{n\_ln} * 0.4 + R_n * 0.6$.

La Figura 30.a muestra que se obtiene le valor óptimo $WS_n = 0$ y la Figura 30.b presenta su representación en el espacio solución con las funciones Pr_{n_ln} y R_n.

6.1.3.8 $\mathbf{w_{pr} = 0.3}$ y $\mathbf{w_r = 0.7}$

La salida inferida por el modelo es (7, 10) para $k = 7$ y $n = 10$, donde:

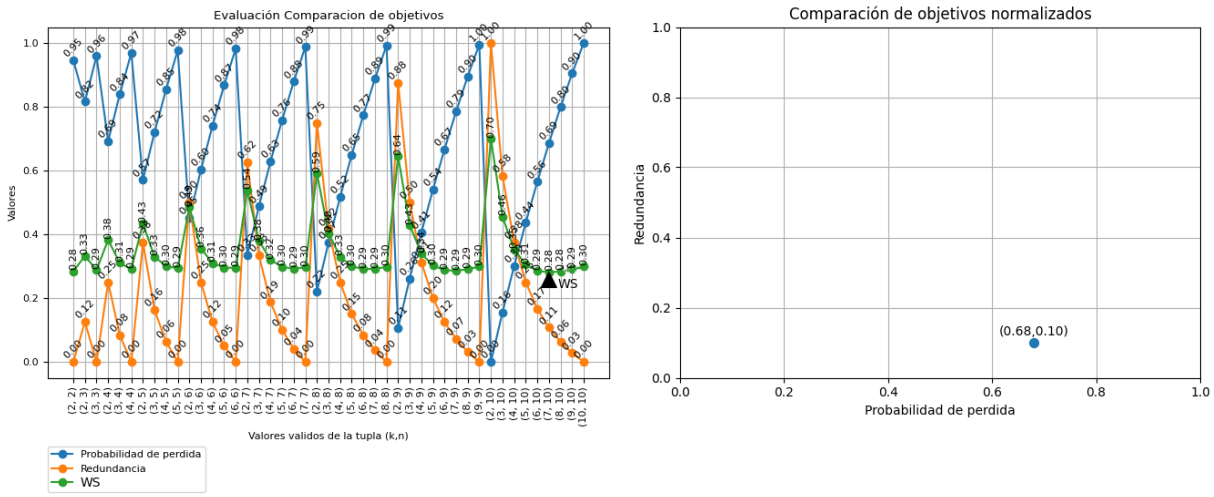
$Pr_{ln}max = -4.660$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -22.72$.

$R_{max} = 10$, $R_{min} = 2$ y $R = 2.86$.

$WS_{max} = 0.70$, $WS_{min} = 0.281$ y $WS = 0.281$.

$Pr_{n\_ln} = \frac{-22.72 + 62.08}{-4.66+62.08} = 0.685$, $R_n = \frac{2.9-2.0}{10.0-2.0} = 0.107$ y $WS_n = \frac{0.281-0.281}{0.700-0.281} = 0.00$.

$Deg(Pr) = \frac{1.354^{e-10}}{1.092e^{-27}} = 1.239e^{+17}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-22.72|} = 2.73$ y $Deg(R) = \frac{2.86}{2.00} = 1.43$.



a) b)

Figura 31. WS $= Pr_{n\_ln} * 0.3 + R_n * 0.7$.

La Figura 31.a muestra que se obtiene le valor óptimo $WS_n = 0$ y la Figura 31.b presenta su representación en el espacio solución con las funciones Pr_{n_ln} y R_n.

6.1.3.9 $\mathbf{w_{Pr} = 0.2}$ y $\mathbf{w_r = 0.8}$

La salida inferida por el modelo es (2, 2) para $k = 2$ y $n = 2$, donde:

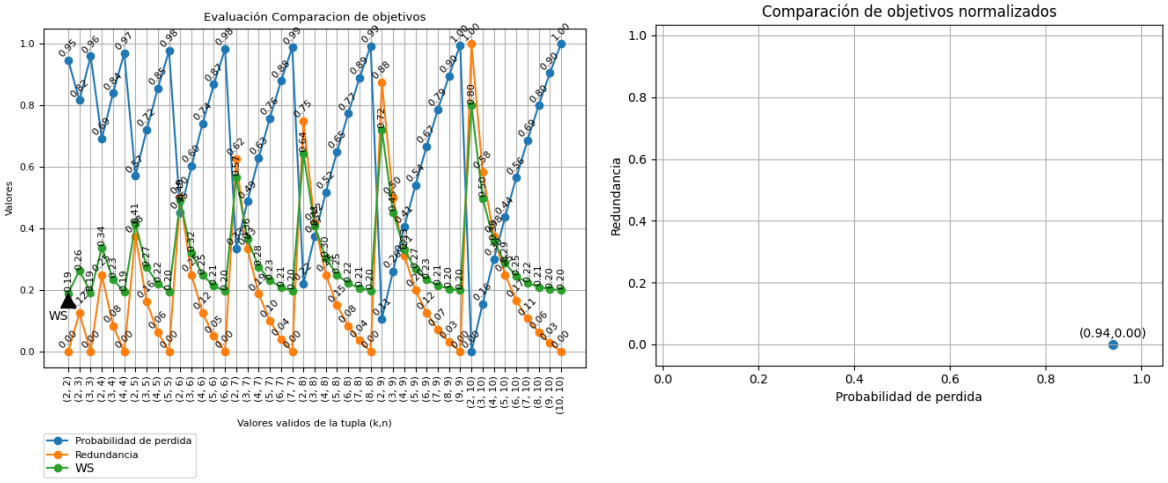
$Pr_{ln}max = -4.660$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -7.764$.

$R_{max} = 10$, $R_{min} = 2$, $R = 2.00$.

$WS_{max} = 0.80$, $WS_{min} = 0.189$ y $WS = 0.189$.

$Pr_{n\_ln} = \frac{-7.764 + 62.08}{-4.66 + 62.08} = 0.946$, $R_n = \frac{2.0-2.0}{10.0-2.0} = 0.00$ y $WS_n = \frac{0.189-0.189}{0.800-0.189} = 0.00$.

$Deg(P_r) = \frac{4.247e^{-04}}{1.092e^{-27}} = 3.888e^{+23}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-7.764|} = 7.99$ y $Deg(R) = \frac{2.00}{2.00} = 1.00$.



a) b)

Figura 32. WS $= Pr_{n\_ln} * 0.2 + R_n * 0.8$.

La Figura 32.a muestra que se obtiene le valor óptimo $WS_n = 0$ y la Figura 32.b presenta su representación en el espacio solución con las funciones Pr_{n_ln} y R_n.

6.1.3.10 $\mathbf{w_{Pr} = 0.1}$ y $\mathbf{w_r = 0.9}$

La salida inferida por el modelo es (2, 2) para $k = 2$ y $n = 2$, donde:

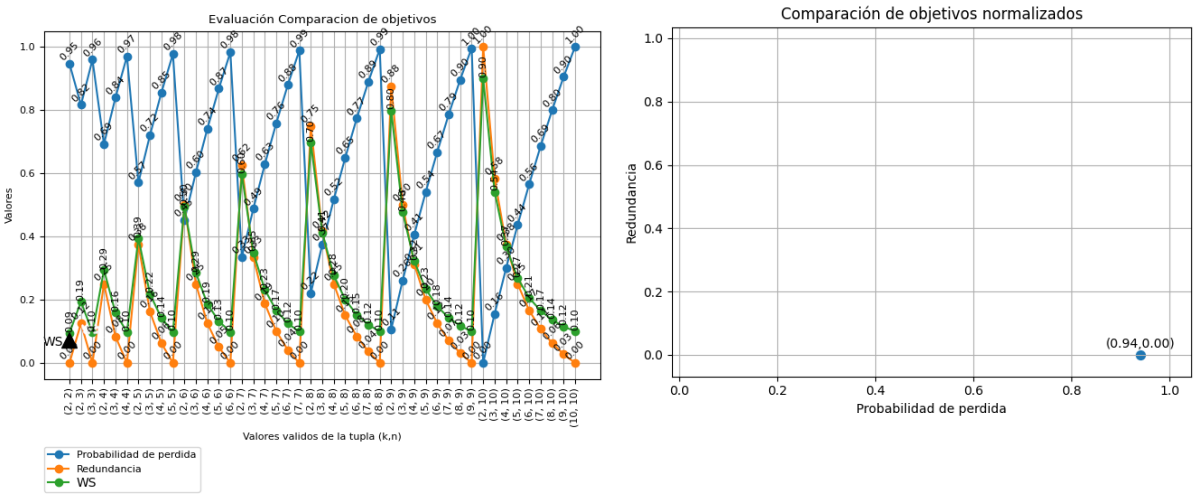
$Pr_{ln}max = -4.660$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -7.764$.

$R_{max} = 10$, $R_{min} = 2$ y $R = 2.00$.

$WS_{max} = 0.90$, $WS_{min} = 0.095$, $WS = 0.095$.

$Pr_{n\_ln} = \frac{-7.764+62.08}{-4.66+62.08} = 0.946$, $R_n = \frac{2.0-2.0}{10.0-2.0} = 0.000$ y $WS_n = \frac{0.095-0.095}{0.900-0.095} = 0.00$.

$Deg(Pr) = \frac{4.247e^{-04}}{1.092e^{-27}} = 3.888e^{+23}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-7.764|} = 7.99$, $Deg(R) = \frac{2.00}{2.00} = 1.00$



a) b)

Figura 33. WS $= Pr_{n\_ln} * 0.1 + R_n * 0.9$.

La Figura 33.a muestra que se obtiene le valor óptimo $WS_n = 0$ y la Figura 32.b presenta su representación en el espacio solución con las funciones Pr_{n_ln} y R_n.

6.1.3.11 $\mathbf{w_{Pr} = 0.0}$ y $\mathbf{w_r = 1.0}$

La salida inferida por el modelo es (2, 2) para $k = 2$ y $n = 2$, donde:

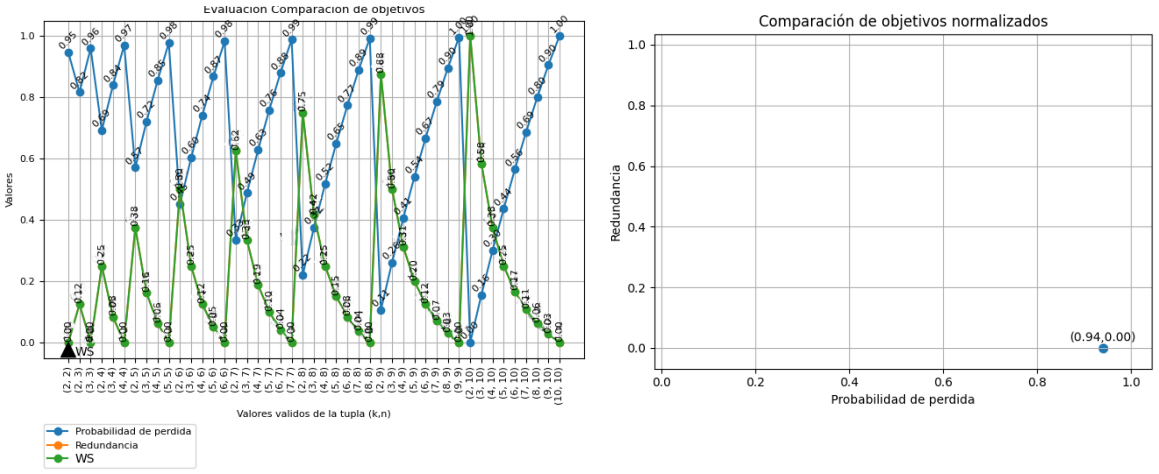
$Pr_{ln}max = -4.660$, $Pr_{ln}min = -62.08$ y $Pr_{ln} = -7.764$

$R_{max} = 10$, $R_{min} = 2$ y $R = 2.00$.

$WS_{max} = 1.00$, $WS_{min} = 0.00$ y $WS = 0.00$.

$Pr_{n\_ln} = \frac{-7.764+62.08}{-4.66+62.08} = 0.946$, $R_n = \frac{2.0-2.0}{10.0-2.0} = 0.00$ y $WS_n = \frac{0.00-0.00}{1.00-0.00} = 0.00$.

$Deg(Pr) = \frac{4.247e^{-04}}{1.092e^{-27}} = 3.888e^{+23}$, $Deg(Pr_{ln}) = \frac{|-62.08|}{|-7.764|} = 7.99$, $Deg(R) = \frac{2.00}{2.00} = 1.00$.



a) b)

Figura 34. $\mathrm{WS} = Pr_{n\_ln} * 0.0 + R_n * 1.0$.

La Figura 34.a muestra que se obtiene el valor óptimo $WS_n = 0$ y la Figura 34.b presenta su representación en el espacio solución con las funciones Pr_{n_ln} y R_n.

6.1.3.12 Frente de soluciones encontrado

Variar los pesos permite obtener diversas soluciones que conforman un frente (véase Figura 35.), sin embargo, las soluciones no se encuentran equidistantes a pesar de que los pesos del vector w son dados de esta manera. Esta situación se puede observar en las Figuras 24.b, 25.b y 26.b, las cuales comparten una misma solución., así como en las Figuras 27.b, 28.b, 33.b y 34.b. Las soluciones únicas que no comparten solución con otros vectores de pesos son las presentadas en las Figuras 29.b, 28.b y 27.b. Esta característica se debe principalmente al método de agregación (reducción del vector objetivo por medio una suma ponderada), con el cual puede entregar soluciones óptimas u subóptimas, pero no tiene la capacidad de representar un frente de Pareto con zonas no convexas.

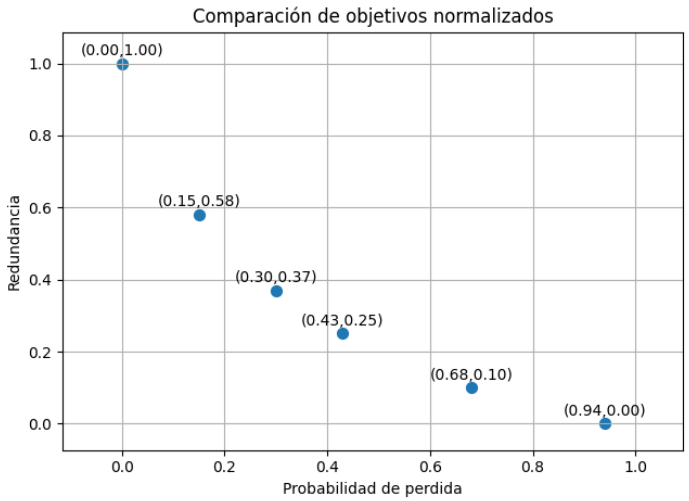


Figura 35. Soluciones encontradas al variar los pesos del vector de preferencias de w que impacta en la función WS (valores normalizados).

Cabe destacar que ninguna solución del frente obtuvo el peor valor posible para Pr_{ln}, estas soluciones son presentadas en las Figuras 29.b, 28.b y 27.b, a pesar de que era posible como sucedió con los valores de preferencia $w_{Pr} = 0.1$ y $w_R = 1.0$ en la sección 6.1.2.3, este problema se debido a que la red para esa instancia fue entrenada con pesos inicializados aleatoriamente y no como en los subproblemas secuenciales que toma los pesos de un vecino cercano para solo refinar la solución.

6.1.4 Análisis estadístico

Los valores de las 30 ejecuciones de la Ptr-Net con instancias de $|N| = 10$ se presentan en las Tabla 2, Tabla 3 y Tabla 4.

En la Tabla 4. Análisis estadístico del parámetro WO y WO_n., WO_n obtiene valores cercanos a cero en las 30 instancias por lo que se encontraron las soluciones óptimas del problema, con distintos valores de preferencias para w. La Tabla 2 muestra la variabilidad de los CSPs con respecto Pr, los valores son similares entre las distintas configuraciones de w, cuando $w_{pr} = 1$ entonces se prioriza Pr y se ignora R resultando en un valor de 0 y varianza 0.

Tabla , cuando w_r se aproxima a los valores de 0 y 1 las soluciones tienden a ser iguales, los valores de la varianza lo muestran. En los casos intermedios, las soluciones tienen valores similares porque los valores de varianza son muy pequeños.

Tabla 2. Análisis estadístico del parámetro Pr_{ln} y Pr_{n_nl}.

Pesos		Pr_{ln}			Pr_{n_nl}		
w_{pr}	w_R	Media	Varianza	Desviación estándar	Media	Varianza	Desviación estándar
1.0	0.0	-60.9309	0.806346	0.8980	0.0000	0.0000000000	0.0000
0.9	0.1	-60.9309	0.806346	0.8980	0.0000	0.0000000000	0.0000
0.8	0.2	-60.9309	0.806346	0.8980	0.0000	0.0000000000	0.0000
0.7	0.3	-52.2001	0.596837	0.7726	0.1549	0.0000004625	0.0007
0.6	0.4	-52.2001	0.596837	0.7726	0.1549	0.0000004625	0.0007
0.5	0.5	-44.0587	0.434893	0.6595	0.2993	0.0000011668	0.0011
0.4	0.6	-36.3937	0.308196	0.5552	0.4353	0.0000015900	0.0013
0.3	0.7	-22.3213	0.131250	0.3623	0.6849	0.0000012674	0.0011
0.2	0.8	-7.3645	0.092442	0.3040	0.9503	0.0000188169	0.0043
0.1	0.9	-7.3647	0.092321	0.3038	0.9503	0.0000187862	0.0043
0.0	1.0	-7.3484	0.102786	0.3206	0.9506	0.0000211564	0.0046

Tabla 3. Análisis estadístico del parámetro R y R_n.

Pesos		R			R_n		
w_{Pr}	w_R	Media	Varianza	Desviación estándar	Media	Varianza	Desviación estándar
1.0	0.0	10.000	0.0000	0.0000	1.0000	0.00000	0.0000
0.9	0.1	10.000	0.0000	0.0000	1.0000	0.00000	0.0000
0.8	0.2	10.000	0.0000	0.0000	1.0000	0.00000	0.0000
0.7	0.3	6.8776	0.0011	0.0335	0.5837	0.00016	0.0127
0.6	0.4	5.1624	0.0017	0.0213	0.5837	0.00026	0.0169
0.5	0.5	4.9934	0.0034	0.0043	0.3750	0.00056	0.0225
0.4	0.6	4.0043	0.0014	0.0419	0.2500	0.00021	0.0145
0.3	0.7	2.8600	0.0012	0.0583	0.1075	0.00013	0.0011
0.2	0.8	2.0000	0.0000	0.0000	0.0000	0.00000	0.0000
0.1	0.9	2.0000	0.0000	0.0000	0.0000	0.00000	0.0000
0.0	1.0	2.0000	0.0000	0.0000	0.0000	0.00000	0.0000

Tabla 4. Análisis estadístico del parámetro WO y WO_n.

Pesos		WO			WO_n		
w_{Pr}	w_R	Media	Varianza	Desviación estándar	Media	Varianza	Desviación estándar
1.0	0.0	0.0000	0.0000000000	0.0000	0.0000	0.0000	0.0000
0.9	0.1	0.1000	0.0000000000	0.0000	0.0000	0.0000	0.0000
0.8	0.2	0.2000	0.0000000000	0.0000	0.0000	0.0000	0.0000
0.7	0.3	0.2835	0.0000002266	0.0005	0.0000	0.0000	0.0000
0.6	0.4	0.3264	0.0000001665	0.0004	0.0000	0.0000	0.0000
0.5	0.5	0.3372	0.0000002917	0.0005	0.0000	0.0000	0.0000
0.4	0.6	0.3241	0.0000002544	0.0005	0.0000	0.0000	0.0000
0.3	0.7	0.2807	0.0000001141	0.0003	0.0000	0.0000	0.0000
0.2	0.8	0.1901	0.0000007527	0.0009	0.0000	0.0000	0.0000
0.1	0.9	0.0950	0.0000001879	0.0004	0.0000	0.0000	0.0000
0.0	1.0	0.0000	0.0000000000	0.0000	0.0000	0.0000	0.0000

6.1.5 Comparación con otros algoritmos

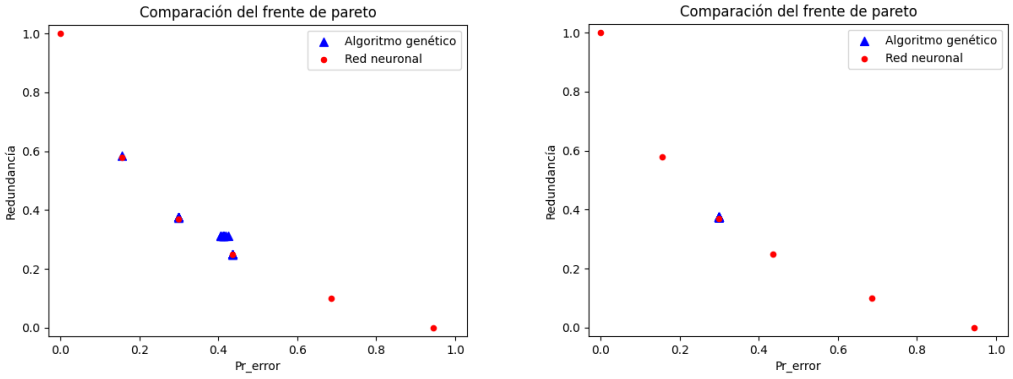
Varias implementaciones de algoritmos genéticos fueron utilizadas para evaluar el desempeño de la red neuronal. En algoritmos genéticos, la calidad de una solución depende fuertemente del tiempo de ejecución, por lo tanto, la comparativa considera variar la cantidad de generaciones. El enfoque analiza la calidad las respuestas entregadas después de cierta cantidad de generaciones con respecto a los resultados proporcionados por la red neuronal.

6.1.5.1 Algoritmo genético simple

El algoritmo simple no está basado en el principio de Pareto, la comparativa punto a punto entre los resultados de la red neuronal y el algoritmo genético se muestran en la Figura 36. La Figura 36.a muestra los resultados de la ejecución después de 500 generaciones con un tiempo de ejecución medio de 0.7921 segundos(s), varianza de 0.0090 y desviación de 0.095. La Figura 36.b presenta los resultados después de 1,000 generaciones con una media de 2.8185 s, varianza de 0.0919 y desviación de 0.3032.

En ambas Figuras (36.a y 36.b) se puede observar que el algoritmo genético tiene problemas de convergencia con pocas generaciones. El incremento en el número de generaciones en el algoritmo genético simple aumenta la posibilidad de encontrar mejores soluciones (vea Figura 37), pero con un tiempo mucho mayor de ejecución. Los valores de la esquina inferior derecha muestran que la red Ptr-Net

está influenciada por soluciones obtenidas previamente y presenta una diferencia con respecto al algoritmo genético.



a) 500 generaciones b) 1,000 generaciones

Figura 36. Soluciones de la red Ptr-Net y del algoritmo genético simple con dos valores de generaciones.

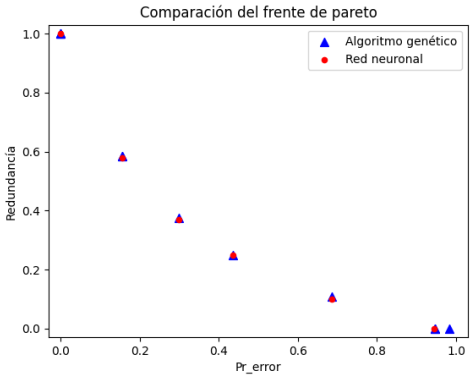


Figura 37. Resultado del algoritmo genético simple con la cantidad máxima de generaciones.

6.1.5.2 NSGA-II y MOCell

Los algoritmos NSGA-II y MOCell están basado en el principio de Pareto. La Tabla 5 presenta los tiempos de ejecución de NSGA-II para varios valores de generaciones. Los resultados de la comparativa entre la red neuronal y NSGA II son presentados en la

Figura 38. En general, se puede apreciar como un mayor número de generaciones incrementa el alcance y la diversidad de las soluciones en el espacio de búsqueda hacía el frente de Pareto. Mientras que el número mínimo de generaciones solo produce una solución de 0.75 de redundancia y 0.22 de Pr_{n_ln}, esto evidencia una solución pobre en comparación con la Pr_Net en referencia al valor obtenido en Pr_{n_ln}, véase

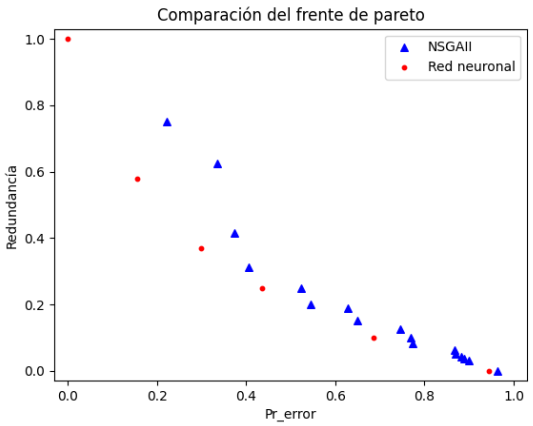
Figura **38**.a, el incremento de estas permite obtener valores con mayor diversidad en todo el frente de Pareto, e incluye las soluciones generadas por la Ptr-Net (véase

Figura **38**.b y

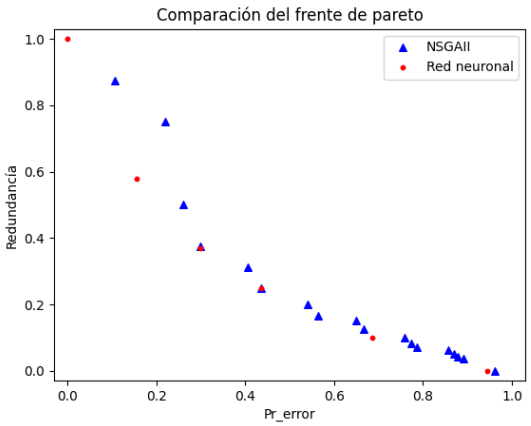
Figura 38.c).

Tabla 5. Tiempos de ejecución del algoritmo NSGA-II con diferente cantidad de generaciones.

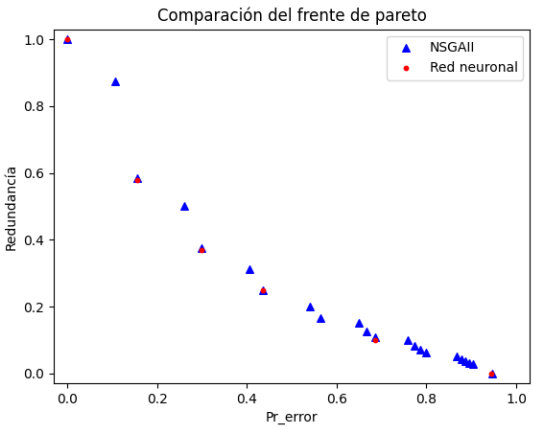
Generaciones máximas	Media (s)	Varianza	Desviación estándar
500	1.82	0.0084	0.090
1,000	4.72	0.0082	0.286
10,000	32.99	2.5273	1.589
15,000	99.17	5.58	2.363



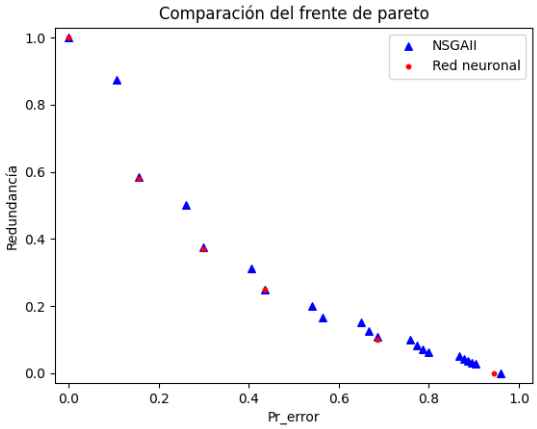
a) 2,500 generaciones.



b) 5,000 generaciones.



c) 10,000 generaciones.



d) 15,000 generaciones.

Figura 38. Resultado de NSGA-II con diferente cantidad de generaciones.

La Tabla 6 presenta los tiempos de ejecución de MOCell ara varias configuraciones de generaciones. Los resultados de la comparativa entre la red Ptr-Net y MOCell son presentados en la

Figura 39 a diferencia de NSGA II, MOCell proporciona soluciones en todo el espacio de búsqueda con el mínimo número de generaciones, véase

Figura **39**.a. El incremento en el número de generaciones no muestra una mejora significativa en la calidad o la distribución en el espacio de búsqueda.

Tabla 6. Tiempos de ejecución del algoritmo MOCell con diferente cantidad de generaciones.

Generaciones máximas	Media (segundos)	Varianza	Desviación estándar
2,500	8.8073	0.0977	0.312
5,000	19.5053	1.1054	1.051
10,000	38.6785	3.9133	1.978
15,000	57.1796	5.7513	2.398

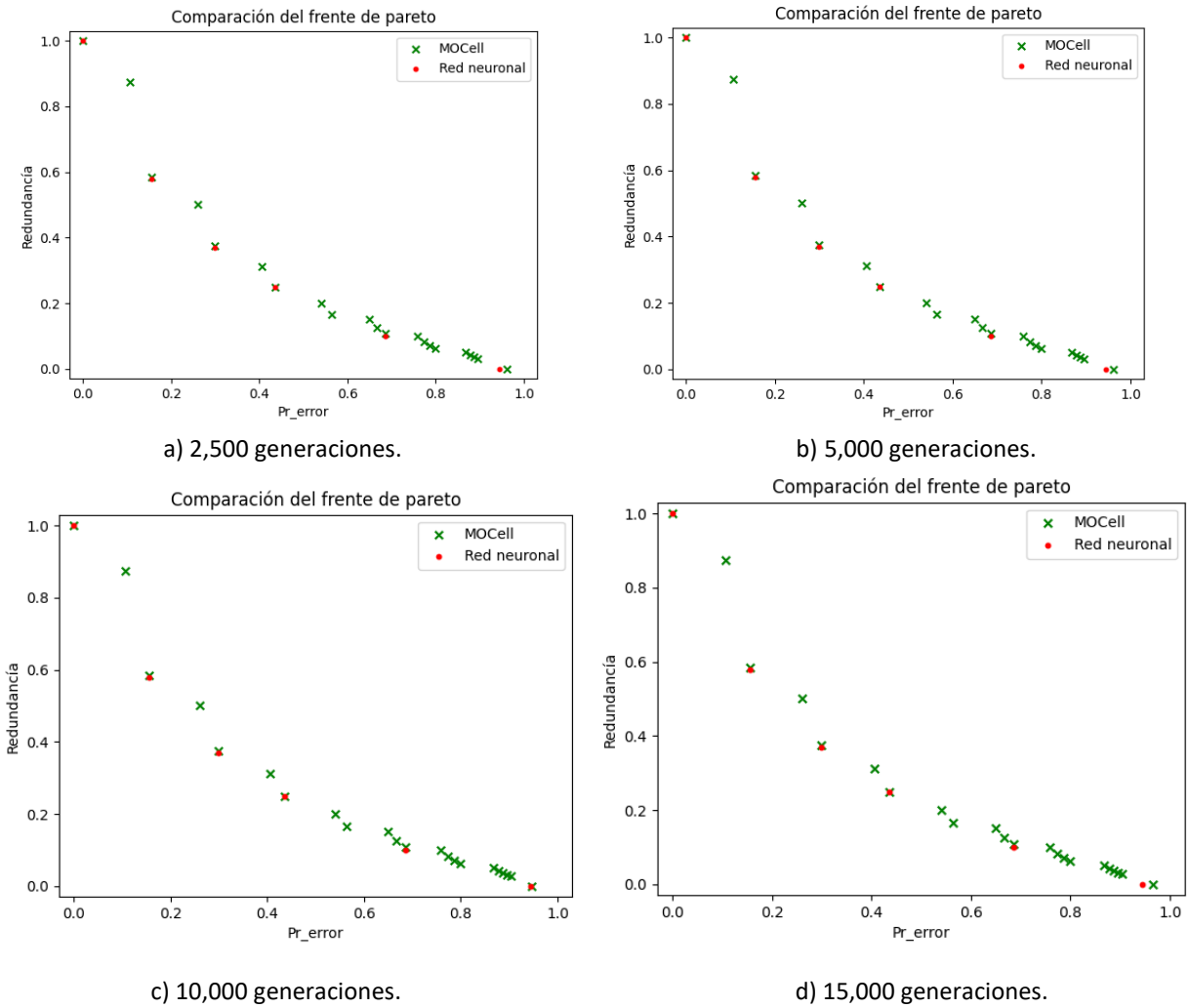


Figura 39. Comparativa de la red neuronal y MOCell con diferente cantidad de generaciones.

Finalmente, se puede aumentar el número de generaciones a 25,000 para evaluar si existe alguna diferencia significativa. Los resultados de esta simulación son presentados en la Figura 40 donde se puede notar que MOCell y NSGA encuentran soluciones con gran diversidad y no tiene problemas para proporcionar soluciones del frente de Pareto. Esta calidad de las soluciones necesita un tiempo considerable, el algoritmo necesita mucho tiempo para que la solución se aproxime al frente de Pareto, como lo muestran las

Figura **38** y

Figura 39, y las Tabla 5 y **Tabla 6**. En contraste, una red Ptr-Net previamente entrenada entrega resultados del frente de Pareto en menor tiempo, con una media: 0.1048 s en CPU (ver Tabla 7), pero no logra replicar la diversidad de los algoritmos genéticos al variar los pesos de vector de preferencias.

El GA simple, NSGA-II y MOCell tiene problemas para encontrar soluciones en la esquina inferior derecha. Una posible explicación de esta situación es la representación del individuo, el segundo cromosoma de la solución tiene una estructura binaria (véase Figura 17). Por este motivo, los operadores de mutación o recombinación tienen dificultad para representar todas las soluciones, por lo que se generan soluciones sub óptimas que encuentran dificultad para aproximarse al frente Pareto.

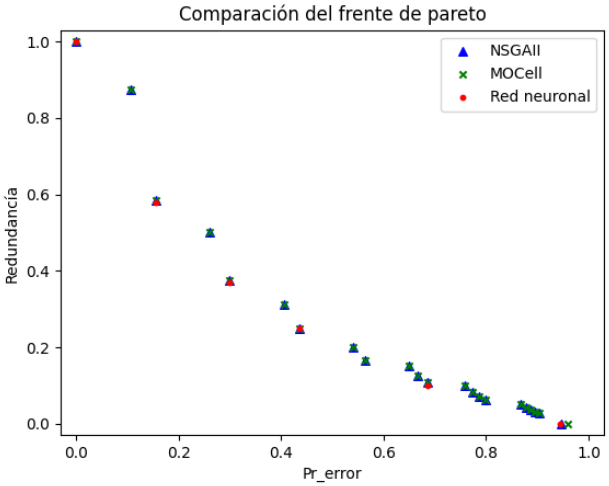


Figura 40. Comparativa de la red Ptr-Net, NSGA II y MOCell con 25,000 generaciones.

6.1.5.3 Ramificación y poda (BandB)

El tiempo de ejecución del algoritmo de ramificación se presenta en la Tabla 7 y los resultados obtenidos son mostrados en la Figura 41. Esta estrategia encuentra soluciones similares a la red aunque ambas técnicas no estén basadas en el frente de Pareto. La influencia de las soluciones previamente encontradas puede asemejarse al frente de Pareto. No obstante, esto no sucedería si el entrenamiento hubiera iniciado con el objetivo de redundancia. Por lo que, un orden lexicográfico puede ser importante para el entrenamiento de la Ptr-Net, con tal de descartar soluciones dominadas, además de evitar partir desde un extremo del espacio de búsqueda.

En la Tabla 7, el tiempo medio de ejecución es de 5.16 segundos, pero con una alta varianza con respecto a la media. Esto se debe principalmente a que el algoritmo de ramificación y poda tiene la capacidad de encontrar una solución de forma inmediata en el árbol o lo puede recorrer en su totalidad.

Tabla 7. Tiempos que de ejecución entre ramificación y poda, heurística y redes neuronales.

Nombre	N (Tamaño de la entrada)	Cantidad de iteraciones	Media (s)	Varianza	Desviación Estándar
Ramificación y poda	10	30	5.16371	12.02042	3.46704
Heurística preordenada	10	30	0.61142	0.018354	0.135477
Ptr-Net CPU	10	30	0.1048	1.3033	0.0036
Ptr-Net GPU	10	30	0.3583	0.0015	0.0388

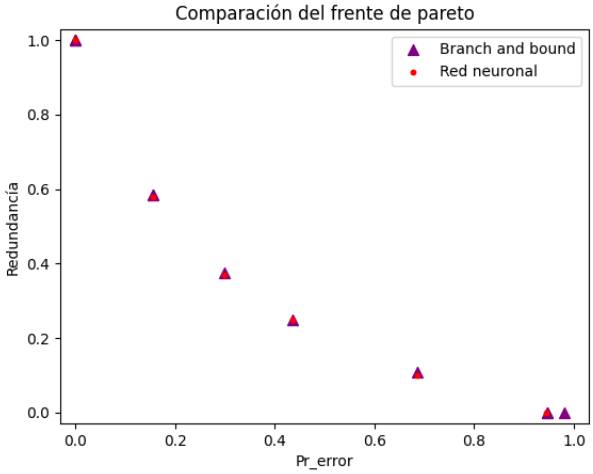


Figura 41. Comparativa de la red neuronal y Ramificación y poda.

Las soluciones encontradas por esta heurística se pueden visualizar en la Figura 42, las soluciones son similares a la Ptr-net pero con varias soluciones dominada en la esquina inferior derecha, este comportamiento se debe a la forma de explorar el espacio de soluciones. Esta heurística aprovecha la estructura básica del problema para no visitar todas las soluciones por lo que se tienen tiempo menores a al GA simple, NSGA-II, MOCell y ramificación y poda. Sin embargo, ligeramente mayores que los tiempos de Ptr-Net (véase Tabla 7). Esta solución es aplicable solo porque hay una forma de ordenar la búsqueda y sacar provecho de ella, pero podría no ser aplicable con un tercer objetivo. La solución dominada también se hace presente en esta técnica, esto se debe a la existencia de múltiples soluciones óptimas para el objetivo redundancia, el algoritmo toma alguna de estas soluciones aunque no necesariamente beneficie al objetivo Pr_{pr_ln}.

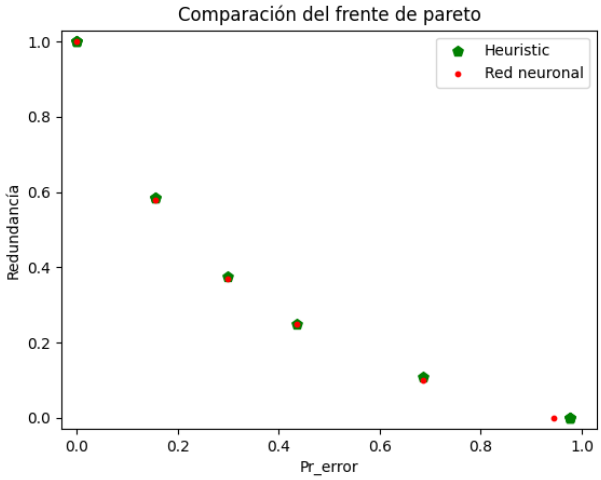


Figura 42. Comparativa de la red neuronal y Heurística preordenado.

Las soluciones encontradas por la Ptr-Net fueron Pareto óptimas, pero se mantenían dentro del espacio convexo de búsqueda. En contraste, el GA con técnicas de dominancia de Pareto mostró una mayor capacidad de generar soluciones fuera de la zona convexa con una gran diversidad de soluciones, una desventaja del AG radica en los altos tiempos de ejecución debido a los operadores estocásticos y la codificación del individuo, ambos intrínsecos a los EA en contraste de la Ptr-Net. Los métodos no basados en dominancia encontraron soluciones similares a la Ptr-Net pero con tiempos mayores, en algunos casos con alta varianza. Además, la Ptr-Net mantiene una influencia de las soluciones resueltas anteriormente que pueden ayudar a evitar soluciones dominadas en los extremos del espacio de solución.

Por último, la Ptr-Net no deja de ser un algoritmo con una distribución categórica al elegir los elementos que conformaran la secuencia, en este caso se comporta como una función determinística debido a que las probabilidades de los elementos de la secuencia óptima son ≈ 1 en concordancia por lo expuesto por Sutton y Barto (2018). Esta propiedad puede no mantenerse si se aumenta N, entregando soluciones de menor calidad, para evitar esto se puede realizar un intercambio tiempo-calidad, para realizar un muestro sobre la distribución con el fin de explorar y explotar soluciones mejores.

Capítulo 7 Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones correspondientes a este trabajo de tesis y se expone futuras direcciones de investigación.

7.1 Conclusiones

En este trabajo se aborda el problema de configurar un sistema AR-RRNS con el uso de redes neuronales de principio a fin (*end-to-end*) con una arquitectura de red apuntadora (en inglés, *pointer network*). La red fue entrenada utilizando un entrenamiento por refuerzo con descenso de gradiente, estos métodos permiten reducir los dos inconvenientes presentes en las redes neuronales para afrontar problemas combinatorios: la falta de datos históricos y la dependencia de un tercer algoritmo para la generación de datos etiquetados.

La red apuntadora busca minimizar dos objetivos fundamentales en los sistemas AR-RRNS: la probabilidad de falla y la redundancia de los archivos, ambos objetivos están en conflicto porque el minimizar uno implica un incremento en el otro. La probabilidad de falla es un elemento crítico que tiene consecuencias graves en la accesibilidad de los archivos. Mientras, la redundancia de los archivos impacta directamente el tamaño de los archivos y, en consecuencia, en otros factores como costos de almacenamiento y tiempos de codificación y decodificación.

Varias estrategias de optimización multi objetivo se pueden utilizar para abordar el problema mencionado. Parte fundamental de esta tesis es mostrar las ventajas, desventajas y limitaciones del enfoque propuesto. La evaluación experimental compara el funcionamiento de la arquitectura propuesta con ramificación y poda y varias versiones de algoritmos evolutivos, además de considera la información de 10 proveedores de nube reales.

La estrategia de ramificación y poda no está basada en población por lo que requiere más tiempo para encontrar soluciones en diferentes partes del espacio de búsqueda y muestra una alta varianza en los tiempos de ejecución en comparación con la red apuntadora. El algoritmo genético simple no está basado en población de Pareto mientras NSGA-II y MOCell si lo están. La red apuntadora tiene tiempos de ejecución menores que los cuatro algoritmos. En cuanto a calidad de las soluciones, la red propuesta

genera resultados comparables a los proporcionados por el algoritmo de ramificación y poda y al algoritmo genético simple, pero no encuentra todas las soluciones de los algoritmos basados en población Pareto.

Las redes neuronales son una alternativa importante y con un futuro prometedor para abordar problemas combinatorios, sobre todo en ambientes dinámicos con cambios rápidos, situaciones donde se busca obtener una solución con tiempo limitado y con cierto margen de error. Actualmente, estas estrategias están en una etapa temprana de desarrollo para competir contra otros enfoques ampliamente probados y utilizados. Sin embargo, su incursión en la optimización multi objetivo ha logrado avances significativos gracias a la atención prestada por parte de la comunidad científica.

7.2 Trabajo a futuro

La revisión de la literatura y el desarrollo del trabajo permitieron identificar diversas direcciones relacionadas con la optimización combinatoria y el aprendizaje por refuerzo. Con la idea de usar este trabajo como base y poder implementar mejoras que se reflejen en su desempeño, se presentan tres futuras direcciones en el área:

- El empleo de funciones de descomposición en lugar de la suma ponderada con el objetivo de obtener puntos de Pareto óptimo en zonas no convexas.

- El desarrollo de una arquitectura basada en "*transformers*", recientemente estas estructuras han demostrado un mejor desempeño que las LSTMs y convolucionales en diversas áreas, este nuevo enfoque puede aportar mejoras significativas en la eficiencia de la red.

- El uso de redes neuronales en conjunto con otra técnica clásica (*long-aside*), el aprendizaje por refuerzo y estas técnicas son alternativas que se están explorando pero requieren un conocimiento especializado en programación matemática, investigación de operaciones, etc. La red solo se encarga de tomar decisiones importantes dentro de los algoritmos, por ejemplo: decidir si se debe o no explorar la rama de un árbol de búsqueda.

Hay una gran oportunidad de extender el trabajo o mejorar el desempeño de la propuesta, sin embargo, el trabajo futuro se enfoca en presenta las direcciones más relevantes en base a la experiencia obtenida durante el desarrollo de esta tesis.

Literatura citada

Alba, E., Dorronsoro, B., Luna, F., Nebro, A. J., Bouvry, P., and Hogie, L. 2007. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. *Computer Communications*, *30*(4), 685–697. https://doi.org/10.1016/j.comcom.2006.08.033

Allenson, R. 1992. *Genetic algorithms with gender for multi-function optimisation*. *Edinburgh Parallel Computing Centre, Edinburgh, Scotland, Tech. Rep. EPCC-SS92-01*

AlZain, M. A., Pardede, E., Soh, B., and Thom, J. A. 2012. Cloud Computing Security: From Single to Multi-clouds. *2012 45th Hawaii International Conference on System Sciences*, 5490–5499. https://doi.org/10.1109/HICSS.2012.153

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. 2010. A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58. https://doi.org/10.1145/1721654.1721672

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38. https://doi.org/10.1109/MSP.2017.2743240

Asmuth, C., and Bloom, J. 1983. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, *29*(2), 208–210. https://doi.org/10.1109/TIT.1983.1056651

Babenko, M., Chervyakov, N., Tchernykh, A., Kucherov, N., Shabalina, M., Vashchenko, I., Radchenko, G., and Murga, D. 2017. Unfairness Correction in P2P Grids Based on Residue Number System of a Special Form. *2017 28th International Workshop on Database and Expert Systems Applications (DEXA)*, *Augus 28-31, 2017, Lyon, France*, 147–151. https://doi.org/10.1109/DEXA.2017.46

Babitha, M. P., and Babu, K. R. R. 2016. Secure cloud storage using AES encryption. *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*, 9-10 Septembe, 2016, Pune, India, 859–864. https://doi.org/10.1109/ICACDOT.2016.7877709

Bäck, T. 1996. *Evolutionary algorithms in theory and practice*. Oxford university press. https://doi.org/10.1093/oso/9780195099713.001.0001

Bahdanau, D., Cho, K., and Bengio, Y. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. *3rd International Conference on Learning Representations, ICLR 2015 San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*

Bello, I., Pham, H., Le, Q. V, Norouzi, M., and Bengio, S. 2016. *Neural Combinatorial Optimization with Reinforcement Learning*. *2*, 1–5. Consultado el 1/10/2020 de https://openreview.net/pdf?id=Bk9mxlSFx

Bessani, A., Correia, M., Quaresma, B., André, F., and Sousa, P. 2013. DepSky: Dependable and Secure

Storage in a Cloud-of-Clouds. *ACM Transactions on Storage*, *9*(4), 1–33. https://doi.org/10.1145/2535929

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. *Springer New York.*

Blakley, G. R. 1979. Safeguarding cryptographic keys. *1979 International Workshop on Managing Requirements Knowledge (MARK)*, 313–318. https://doi.org/10.1109/MARK.1979.8817296

Brassard, G., and Bratley, P. 1996. *Fundamentals of Algorithmics* (I. Zucker (ed.)). Prentice-Hall International.

Burke, E. K., and Graham, K. 2014. Search methodologies: Introductory tutorials in optimization and decision support techniques, second edition. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques,*https://doi.org/10.1007/978-1-4614-6940-7

Cachin, C., Keidar, I., and Shraer, A. 2009. Trusting the cloud. *ACM SIGACT News*, *40*(2), 81–86. https://doi.org/10.1145/1556154.1556173

Celesti, A., Fazio, M., Villari, M., and Puliafito, A. 2016. Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems. *Journal of Network and Computer Applications*, *59*, 208–218. https://doi.org/10.1016/j.jnca.2014.09.021

Charnes, A., and Cooper, W. W. 1957. Management Models and Industrial Applications of Linear Programming. *Management Science*, *4*(1), 38–91. https://doi.org/10.1287/mnsc.4.1.38

Chervyakov, N., Babenko, M., Tchernykh, A., Kucherov, N., Miranda-López, V., and Cortés-Mendoza, J. M. 2019. AR-RRNS: Configurable reliable distributed data storage systems for Internet of Things to ensure security. *Future Generation Computer Systems*, *92*, 1080–1092. https://doi.org/10.1016/j.future.2017.09.061

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*,October 25-29, 2014, Doha, Qatar, 1724–1734. https://doi.org/10.3115/v1/D14-1179

Cloud Security Alliance. 2020. Top Threats to Cloud Computing: Egregious Eleven Deep Dive. *Cloud Security Alliance*, 1–30. Consultado el 12/01/2021 de https://cloudsecurityalliance.org/artifacts/top-threats-egregious-11-deep-dive/

CloudHarmony. 2015. Consultado el 03/2018 de https://cloudharmony.com/status

Coello-Coello, C. A. 1999. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, *1*(3), 269–308. https://doi.org/10.1007/BF03325101

Coello-Coello, C. A., Lamont, G. B., and Veldhuizen, D. A. Van. (2007). *Evolutionary Algorithms for Solving*

Multi-Objective Problems (D. E. Goldberg and J. R. Koza (Eds.); second). Springer International Publishing.

Cohon, J. L., and Marks, D. H. 1975. A review and evaluation of multiobjective programing techniques. *Water Resources Research*, *11*(2), 208–220. https://doi.org/10.1029/WR011i002p00208

Daemen, J., and Rijmen, V. 2002. The Design of Rijndael. En *New York*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-04722-4

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. https://doi.org/10.1109/4235.996017

Dozat, T. 2016. Incorporating Nesterov Momentum into Adam. *International Conference on Learning Representations Workshop*, May 2 - 4, 2016, San Juan, Puerto Rico.

Duchi, J., Hazan, E., and Singer, Y. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, *12*(61), 2121–2159.

Ermakova, T., and Fabian, B. 2013. Secret Sharing for Health Data in Multi-provider Clouds. *2013 IEEE 15th Conference on Business Informatics*, July 15 - 18, 2013, Vienna, Austria, 93–100. https://doi.org/10.1109/CBI.2013.22

García-Hernández, L. E. 2019. *Optimización multi-objetivo de un sistema de almacenamiento de datos en la multi-nube con un esquema de compartición de secretos basado en RRNS*. Tesis de maestria en ciencias. Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California. 99 pp.

Garner, H. L. 1959. The Residue Number System. *IEEE Transactions on Electronic Computers*, *EC-8*(2), 140–147. https://doi.org/10.1109/TEC.1959.5219515

Glorot, X., and Yoshua, B. 2010. Understanding the difficulty of training deep feedforward neural networks Xavier. *Proceedings of the thirteenth international conference on artificial intelligence and statistics.JMLR Workshop and Conference Proceedings*, AISTATS 2010, Sardinia, Italy, May 13-15, 2010. 1074–1078.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machilne Learning*. Addiso-Wesley Publishing Company, Inc.

Goodfellow, I., Bengio, Y., and Aaron, C. 2016. *Deep Learning* (T. Dietterich (Ed.)). The MIT Press, Cambridge, Massachusetts, USA.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. 2014. Generative Adversarial Networks. *Communications of the ACM*, *63*(11), 139–144. https://doi.org/10.1145/3422622

Graves, A., Wayne, G., and Danihelka, I. 2014. Neural Turing Machines. *arXiv Neural and evolutionary computing*, 1–26. consultado 3/03/21 http://arxiv.org/abs/1410.5401

Gu, S., Chen, X., Zeng, W., and Wang, X. 2018. A deep learning tennis ball collection robot and the implementation on NVIDIA jetson TX1 board. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, *July 9-12*, Auckland, New Zealand, 170–175. https://doi.org/10.1109/AIM.2018.8452263

He, K., Zhang, X., Ren, S., and Sun, J. 2015a. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*,27-30 June, Las Vegas, NV, USA, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

He, K., Zhang, X., Ren, S., and Sun, J. 2015b. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, December 7-13, *2015, Santiago, Chile, Inter*, 1026–1034. https://doi.org/10.1109/ICCV.2015.123

Hinton , G. E. , McClelland, J. L., and Rumelhart, D. E. 1986. Distributed representations. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1 (pp. 77–109).

Hinton, G. E., Osindero, S., and Teh, Y.-W. 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, *18*(7), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

Hochreiter, S., and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hopfield, J. J., and Tank, D. W. 1985 . Neural computation of decisions in optimization problems. *Biological Cybernetics*, *52*(3), 141–152. https://doi.org/10.1007/BF00339943

Hu, H., Zhang, X., Yan, X., Wang, L., and Xu, Y. 2017 . *Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method*. consultado en 23/02/2021 de http://arxiv.org/abs/1708.05930

Kachele, S., Spann, C., Hauck, F. J., and Domaschka, J. 2013 . Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking. *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, december 9-12, 2013, Dresden, Germany, 75–82. https://doi.org/10.1109/UCC.2013.28

Khoufi, I., Laouiti, A., and Adjih, C. 2019 . A survey of recent extended variants of the traveling salesman and vehicle routing problems for unmanned aerial vehicles. *Drones*, *3*(3), 1–30. https://doi.org/10.3390/drones3030066

Kingma, D. P., and Ba, J. 2014 . Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–15 May, San Diego, CA, USA, May 7-9, 2015.

Kingma, D. P., and Welling, M. 2014 . Auto-encoding variational bayes. *2nd International Conference on*

Learning Representations, ICLR 2014 - Conference Track Proceedings, Banff, AB, Canada, April 14-16, 2014.

Kohonen, T. 1982 . Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *43*(1), 59–69. https://doi.org/10.1007/BF00337288

Krawczyk, H. 1994 . Secret Sharing Made Short. In 13th Annual International Cryptology Conference — *CRYPTO' 93: Vol. 773 LNCS* (pp. 136–146), Santa Barbara, California, USA, August 22-26, 1993. Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-48329-2_12

LeCun, Y. 1989. Generalization and network design strategies. The CRG technical Report secretary Department of computer science, University of Toronto, Toronto, Canada. Technical Report. consultado el 23/03/2021 de http://yann.lecun.com/exdb/publis/pdf/lecun-89.pdf

Li, K., Zhang, T., and Wang, R. 2021 . Deep Reinforcement Learning for Multiobjective Optimization. *IEEE Transactions on Cybernetics*, *51*(6), 3103–3114. https://doi.org/10.1109/TCYB.2020.2977661

Marium, S., Nazir, Q., Shaikh, A. A., Ahthasham, S., and Mehmood, M. A. 2012 . Implementation of Eap with RSA for Enhancing The Security of Cloud Computing. *International Journal of Basic and Applied Sciences*, *1*(3), 177–183. https://doi.org/10.14419/ijbas.v1i3.94

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. 2011 . Cloud computing — The business perspective. *Decision Support Systems*, *51*(1), 176–189. https://doi.org/10.1016/j.dss.2010.12.006

Mell, P., and Grance, T. 2011 . The NIST Definition of Cloud Computing. *National*, *NIST Special Publication 800-145*.

Mignotte, M. 1979 . How to Share a Secret. En *Cryptography* (Vol. 22, Número 11, pp. 371–375). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-39466-4_27

Minsky, M., & Papert, S. 1969. Perceptrons: An introduction to computational geometry. MIT Press. https://doi.org/10.7551/mitpress/11301.001.0001

Miranda-López, V., Tchernykh, A., Cortés-Mendoza, J. M., Babenko, M., Radchenko, G., Nesmachnow, S., and Du, Z. 2018 . Experimental Analysis of Secret Sharing Schemes for Cloud Storage Based on RNS. En *Communications in Computer and Information Science* (Vol. 796, pp. 370–383). https://doi.org/10.1007/978-3-319-73353-1_26

Mohan, P. V. A. 2016 . Residue Number Systems. En *Intelligent Systems Reference Library* (Vol. 66). Springer International Publishing. https://doi.org/10.1007/978-3-319-41385-3

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., and Alba, E. 2014 . MOCell: A Cellular Genetic Algorithm for Multiobjective Optimization. *Nature Inspired Cooperative Strategies for Optimization*, *24*(7), 726–746. https://doi.org/10.1002/INT.V24:7

Nesterov, Y. 1983 . Yurii Nesterov. A method for unconstrained convex minimization problem with the

rate of convergence o(1/k2). *Doklady AN USSR*, *269*, 543–547.

Osyczka, A. 1985 . Multicriteria optimization for engineering design. En J. S. Gero (Ed.), *Design Optimization,* (1a ed., pp. 193–227). Academic Press. https://doi.org/10.1016/B978-0-12-280910-1.50012-X

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Chintala, S. 2019 . PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems* 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. 8026-8037.

Pundkar, S. N., and Shekokar, N. 2016 . Cloud computing security in multi-clouds using Shamir's secret sharing scheme. *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 3-5 March, 2016, Chennai, India, 392–395. https://doi.org/10.1109/ICEEOT.2016.7755427

Qian, L., Luo, Z., Du, Y., and Guo, L. 2009 . Cloud Computing: An Overview. En *IEEE International Conference on Cloud Computing* (pp. 626–631). https://doi.org/10.1007/978-3-642-10665-1_63

Qian, N. 1999 . On the momentum term in gradient descent learning algorithms. *Neural Networks*, *12*(1), 145–151. https://doi.org/10.1016/S0893-6080(98)00116-6

Quisquater, M., Preneel, B., and Vandewalle, J. 2002 . On the Security of the Threshold Scheme Based on the Chinese Remainder Theorem. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 2274, pp. 199–210). https://doi.org/10.1007/3-540-45664-3_14

Rabin, M. O. 1989 . Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, *36*(2), 335–348. https://doi.org/10.1145/62044.62050

Rathanam, G. J., and Sumalatha, M. R. 2014 . Dynamic secure storage system in cloud services. *2014 International Conference on Recent Trends in Information Technology*, April 10-12, 2014, Chennai, India, 1–5. https://doi.org/10.1109/ICRTIT.2014.6996175

Ronneberger, O., Fischer, P., and Brox, T. 2015 . U-Net: Convolutional Networks for Biomedical Image Segmentation. En N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi (Eds.), *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 9351, Número Cvd, pp. 234–241). Springer International Publishing. https://doi.org/10.1007/978-3-319-24574-4_28

Rosenblatt, F. 1957 . The Perceptron. A Perceiving and Recognizing Automaton, *Cornell Aeronautical Laboratory.* Technical Report. Consultado el 21/01/2021 de https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf

Ruder, S. 2016 . *An overview of gradient descent optimization algorithms*. 1–14. consultado 05/02/21

http://arxiv.org/abs/1609.04747

Rudolph, G. 1994 . Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, *5*(1), 96–101. https://doi.org/10.1109/72.265964

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986 . Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Saxe, A. M., Mcclelland, J. L., and Ganguli, S. 2014 . Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *In ICLR 2014 : International Conference on Learning Representations (ICLR) 2014, 14-16 April, Banff,* AB, *Canada*.

Schaffer, J. D. 1985 . Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the first international conference on genetic algorithms and their applications*, July 24-26, 1985, at the Carnegie-Mellon University, Pittsburg, PA. Lawrence Erlbaum Associates. Inc., Publishers.

Shamir, A. 1979 . How to share a secret. *Communications of the ACM*, *22*(11), 612–613. https://doi.org/10.1145/359168.359176

Smith, K. A. 1999 . Neural networks for combinatorial optimization: A review of more than a decade of research. *INFORMS Journal on Computing*, *11*(1), 15–34. https://doi.org/10.1287/ijoc.11.1.15

Sutskever, I., Vinyals, O., and Le, Q. V. 2014 . Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems 27*, *4*(January), 3104–3112.

Sutton, R. S., and Barto, A. G. 2018 . *Reinforce learning: An introduction* (second). The MIT Press, London, England.

Tchernykh, A., Babenko, M., Chervyakov, N., Miranda-López, V., Kuchukov, V., Cortés-Mendoza, J. M., Deryabin, M., Kucherov, N., Radchenko, G., and Avetisyan, A. 2018 . AC-RRNS: Anti-collusion secured data sharing scheme for cloud storage. *International Journal of Approximate Reasoning*, *102*(July), 60–73. https://doi.org/10.1016/j.ijar.2018.07.010

Tchernykh, A., Babenko, M., Miranda-Lopez, V., Drozdov, A. Y., and Avetisyan, A. 2018 . WA-RRNS: Reliable Data Storage System Based on Multi-cloud. *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 21-25, 2018, Vancouver, British Columbia, Canada, 666–673. https://doi.org/10.1109/IPDPSW.2018.00107

Tchernykh, A., Schwiegelsohn, U., Talbi, E. ghazali, and Babenko, M. 2019 . Towards understanding uncertainty in cloud computing with risks of confidentiality, integrity, and availability. *Journal of Computational Science*, *36*. https://doi.org/10.1016/j.jocs.2016.11.011

Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. 2008 . A break in the clouds. *ACM SIGCOMM Computer Communication Review*, *39*(1), 50–55. https://doi.org/10.1145/1496091.1496100

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. 2017 . Attention Is All You Need. In *Advances in Neural Information Processing Systems, December 4-9, 2017, Long Beach, CA, USA,* (pp. 5998–6008).

Veldhuizen, D. A. Van. 1999 . *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. Tesis de doctorado en ciencias. Faculty of the Graduate School of Engineering of the Air Force Institute of Technology. 87 pp. consultado el 16/03/21 de https://apps.dtic.mil/sti/pdfs/ADA364478.pdf

Vinyals, O., Fortunato, M., and Jaitly, N. 2015 . Pointer Networks. In NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, December 7-12, 2015, Montreal, Quebec, Canadam, 2692–2700.

Vukolić, M. 2010. The byzantine empire in the intercloud. *ACM SIGACT News*, *41*(3), 105–111. https://doi.org/10.1145/1855118.1855137

Widrow, B., and Hoff, M. E. 1960. Adaptative Switching Circuits. In *Wescon Conference Record*, August 23-26, 1960, Los Angeles, Calif., USA. https://doi.org/10.21236/AD0241531

Zeiler, M. D. 2012. *ADADELTA: An Adaptive Learning Rate Method*. Consultado el 23/03/2021 https://arxiv.org/abs/1212.5701

Zissis, D., and Lekkas, D. 2012. Addressing cloud computing security issues. *Future Generation Computer Systems*, *28*(3), 583–592. https://doi.org/10.1016/j.future.2010.12.006