MODELOS DE APRENDIZAJE PROFUNDO PARA EL ANÁLISIS DE TOLERANCIA A FALLAS EN SISTEMAS HPC

DIEGO FERNANDO ACOSTA ORTIZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE FÍSICA
BUCARAMANGA

2024

MODELOS DE APRENDIZAJE PROFUNDO PARA EL ANÁLISIS DE TOLERANCIA A FALLAS EN SISTEMAS HPC

DIEGO FERNANDO ACOSTA ORTIZ

Tesis de maestría para el programa de posgrado de:
Maestría en Matemática Aplicada

Director:
Carlos Jaime Barrios Hernández, Ph. D.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE FÍSICA
BUCARAMANGA
2024

DEDICATORIA

A mis padres y hermana.

AGRADECIMIENTOS

En primer lugar a mis padres por su apoyo incondicional, por ser un faro de luz en los momentos más oscuros.

Al profesor Carlos Jaime Barrios por su labor de mentor, por su apoyo, paciencia y dedicación como director del proyecto.

A Luis Alejandro Torres y en general al equipo que gestiona y administra la plataforma de supercomputación SC3-UIS, por brindarme asesoría, facilitarme datos y recursos de cómputo indispensables para el desarrollo de este proyecto.

Al Laboratorio Nacional de Computación de Alto Rendimiento de Chile por compartirnos datos de la operación de la infraestructura para probar el modelo propuesto.

A mis compañeros del grupo de investigación CAGE.

A la planta docente de la Universidad Industrial de Santander que a través de los años acompañaron mi proceso educativo durante el pregrado y ahora en el posgrado.

A mis amigos Camilo y Daniel quienes siempre me motivaron a no desistir de este proceso académico.

Índice general

Pág

INTRODUCCIÓN **11**

1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA **13**

2. OBJETIVOS **15**

3. MARCO TEÓRICO **16**

3.1. Fundamentos matemáticos aplicados 16

3.1.1. Aprendizaje Automático 16

3.1.2. Redes Neuronales Artificiales ANN 18

3.1.3. Codificadores automáticos o *Autoencoders* 20

3.1.4. Autoencoder Variacional 21

3.2. Conceptos relacionados con HPC 25

3.2.1. Tolerancia a Fallos en HPC 25

3.2.2. Mecanismos de tolerancia a fallos 25

3.2.3. Caracterización de fallos 28

3.3. Métricas 28

3.3.1. Precisión 28

3.3.2. Recall 29

3.3.3. F1 score 29

4. ESTADO DEL ARTE **31**

5. METODOLOGÍA **36**

6. MODELADO **39**

6.1. Datos de entrada 39
6.1.1. Pre-procesamiento 39
6.1.2. Extracción y representación de Características 41
6.2. Arquitectura utilizada 42
6.2.1. Implementación de arquitectura *Autoencoder* 42
6.3. Entrenamiento del modelo 44

7. EVALUACIÓN Y RESULTADOS 45
7.1. Reportes 46

8. CONCLUSIONES Y RECOMENDACIONES 53
8.1. Conclusiones 53
8.2. Recomendaciones 54

BIBLIOGRAFÍA 56

Índice de figuras

Pág

Figura 1. Tipos de ajuste 17
Figura 2. Perceptron 18
Figura 3. Red neuronal no-profunda 19
Figura 4. Red Neuronal Profunda 19
Figura 5. Ejemplo autoencoder 20
Figura 6. Autoencoder variacional 22
Figura 7. Tácticas de disponibilidad. 26
Figura 8. Diagrama CRISP-DM 36
Figura 9. Diagrama SEMMA 37
Figura 10.VAE 42
Figura 11.Encoder 43
Figura 12.Decoder 43
Figura 13.Minimización ELBO 45
Figura 14.Visualización latente predicha 94 % 47
Figura 15.Visualización anotada 48
Figura 16.Métricas reportadas 49
Figura 17.Visualización latente predicha 97 % 50
Figura 18.Visualización latente anotada 51

Índice de cuadros

Pág

Tabla 1. Métricas 1 46
Tabla 2. Métricas 2 49

RESUMEN

TÍTULO: MODELOS DE APRENDIZAJE PROFUNDO PARA EL ANÁLISIS DE TOLERANCIA A FALLAS EN SISTEMAS HPC. *

AUTOR: DIEGO FERNANDO ACOSTA ORTIZ **

PALABRAS CLAVE: HPC, Fault Tolerance, Resilience, Deep Learning.

DESCRIPCIÓN: Las aplicaciones ejecutadas en plataformas de HPC necesitan ser protegidas contra fallos. A medida que los sistemas computacionales se vuelven más grandes, la susceptibilidad a ser afectados por errores también crece. Por lo tanto, la necesidad de proporcionar resiliencia a los sistemas es crítica, y la predicción de fallos se ha convertido en uno de los temas más conocidos.

Obtener información sobre el estado actual de un sistema podría ayudar a establecer algunas estrategias para apoyar la gestión de este tipo de plataformas. Este tema ha sido ampliamente estudiado. Sin embargo, de manera más analítica, hoy en día, dado el creciente volumen de datos de registros que se están generando, han surgido nuevos enfoques en el horizonte.

Varias técnicas proporcionan resiliencia; aquí nos enfocamos en aprovechar los datos de registros para detectar patrones que podrían ofrecer protección adicional contra fallos, brindando mayor fiabilidad.

El uso de técnicas de aprendizaje profundo en los últimos años ha mostrado una mejora sobresaliente en la automatización de múltiples tareas. El monitoreo de plataformas HPC genera enormes cantidades de registros que pueden aprovecharse para descubrir patrones que no pueden ser observados con un análisis sencillo.

---

* Trabajo de investigación

** Facultad de Ciencias. Escuela de Física. Director: Carlos Jaime Barrios Hernández, Ph.D.

ABSTRACT

TITLE: DEEP LEARNING MODELS FOR FAULT TOLERANCE ANALYSIS IN HPC SYSTEMS. *

AUTHOR: DIEGO FERNANDO ACOSTA ORTIZ **

KEYWORDS: HPC, Fault Tolerance, Resilience, Deep Learning.

DESCRIPTION: Applications executed in HPC platforms need to be protected against failure. As long as the computational systems become huge, the susceptibility to be affected by errors grows too, then, the necessity to provide resilience to the systems is critical, and failure prediction has become one of the most well-known issues.

Getting information about the current state of a system could help establish some strategies to support the management of these types of platforms. This topic has been widely studied. However, more analytically, nowadays, given the increased volume of log data that currently are being generated, new approaches have arisen on the horizon.

Several techniques provide resilience; here, we focus on taking advantage of log data to detect patterns that could give extra protection against occurring failures, bringing higher reliability.

The use of DL techniques in recent years has shown an outstanding improvement in automatizing multiple tasks. Monitoring HPC creates vast amounts of log data that can be taken advantage of to discover patterns that can not be seen with a naive analysis.

---

* Research work

** Faculty of Science. School of Physics. Advisor: Carlos Jaime Barrios Hernández, PhD

INTRODUCCIÓN

El eterno debate alrededor de las matemáticas es que si estas son un descubrimiento humano o si, por el contrario, son un producto la mente humanas sigue vigente hoy día, pero su impacto en el desarrollo tecnológico es indiscutible. La agricultura, la navegación, el comercio, entre otras tantas disciplinas usan las matemáticas para realizar cálculos, estimaciones, llevar registros, etcétera, pero bien se sabe que los humanos tenemos una capacidad limitada para realizar gran número de operaciones, por eso se crearon dispositivos como el ábaco, que solventan esta limitación.

En nuestra era contemporánea los ábacos han sido reemplazados por dispositivos más complejos como los computadores, llevando a la ciencia y la ingeniería a alcanzar niveles inimaginables y dando lugar a nuevas disciplinas como la ciencia de datos y la inteligencia artificial (IA), generando un efecto tipo bola de nieve, en el sentido de que cada vez es necesaria más capacidad y rendimiento de cómputo y en donde potenciar los sistemas de cómputo se convierte en un reto en sí, en respuesta a esto, se han desarrollado complejos sistemas de Computación de Alto Rendimiento, también conocidos como supercomputadores o sistemas HPC.

Los modelos de aprendizaje profundo, basados en matemáticas aplicadas, son herramientas cruciales para resolver problemas complejos en ciencia y tecnología. Inspirados en el funcionamiento del cerebro humano, estos modelos permiten la creación de sistemas que pueden aprender y generalizar a partir de datos. En el contexto de la Computación de Alto Rendimiento (HPC), estos modelos son especialmente útiles para la detección de anomalías en la operación de los sistemas.

Identificar fallos y anomalías en sistemas HPC es un desafío crítico, ya que los errores pueden tener consecuencias significativas en aplicaciones sensibles como la medicina y la seguridad. Las técnicas de IA, específicamente los modelos de aprendizaje profundo, facilitan la identificación de patrones además de la detección de anomalías a través del uso de el gran volúmenes de datos que genera la constante operación de un sistema HPC.

A continuación se presenta la estructura del documento: El capítulo 1 formaliza el planteamiento del problema y su justificación para luego pasar a la definición de los objetivos en el 2. En el capítulo 3 se brinda contexto para entender mejor el proyecto planteado. En el capitulo 4 se presenta la revisión de la literatura al rededor del problema y las técnicas utilizadas en el proyecto. En el capítulo 5 se habla de la metodología utilizada para el desarrollo del trabajo de grado. En el capítulo de 6 se encuentra el desarrollo del proyecto y para luego cerrar con los capítulos 7 de resultados y el 8 de conclusiones.

1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA

Los sistemas de Computación de Alto Rendimiento o HPC por sus siglas en inglés, como cualquier sistema, son sistemas susceptibles a fallos, los cuales incluso pueden ser simples cambios en un bit que logran causar que un proceso o una aplicación fallen.[1] En la medida que una plataforma HPC crece en tamaño, la necesidad de proveer resiliencia al sistema es crítica y la estimación de ocurrencias de fallos se convierte en uno de los inconvenientes más conocidos.[2] Por ejemplo, plataformas como la de Supercomputación y Cálculo Científico de la Universidad Industrial de Santander, SC3-UIS [3], en Colombia o el Laboratorio Nacional de Computación de Alto Rendimiento de Chile[4] el monitoreo es complejo porque cuentan con múltiples nodos, decenas de usuarios y trabajos en el ejecución.

Al obtener información del estado de un sistema, se pueden establecer estrategias para soportar la administración de plataformas HPC, existen varios modelos analíticos que sirven para modelar el comportamiento de estos sistemas[5], pero el uso de técnicas de inteligencia artificial, particularmente aprendizaje profundo, puede contribuir a la automatización, disminuyendo la intervención y atención humana en este tipo de plataformas.

---

[1] Bianca Schroeder y Garth A. Gibson. «A large-scale study of failures in high-performance computing systems». En: *Proceedings of the International Conference on Dependable Systems and Networks* 2006 (2006), págs. 249-258. DOI: 10.1109/DSN.2006.5.

[2] Stijn Heldens et al. «The Landscape of Exascale Research: A Data-Driven Literature Analysis». En: *ACM Computing Surveys* 53.2 (2020). DOI: 10.1145/3372390.

[3] Universidad Industrial de Santander. *Supercomputación y Cálculo Científico*. URL: https://www.sc3.uis.edu.co/ (visitado 14-02-2024).

[4] National Laboratory for High Performance Computing Chile. *Laboratorio Nacional de Computación de Alto Rendimiento (NLHPC)*. URL: https://www.nlhpc.cl/ (visitado 14-02-2024).

[5] Ana Gainaru et al. «Failure prediction for HPC systems and applications: Current situation and open issues». En: *International Journal of High Performance Computing Applications* 27.3 (2013), págs. 273-282. DOI: 10.1177/1094342013488258.

De acuerdo con lo anterior, se establece la pregunta de investigación que permitirá guiar al proyecto durante su desarrollo:

> ¿De qué manera se pueden aplicar los algoritmos de aprendizaje profundo para brindar soporte a la administración de plataformas de computación de alto rendimiento?

Este proyecto se enfocará principalmente en el modelamiento matemático que permita implementar un mecanismo de detección de anomalías para brindar tolerancia a fallos esenciales en plataformas HPC.

Palabras clave: Aprendizaje automático, aprendizaje profundo, análisis de datos, matemática aplicada, HPC, tolerancia a fallos, detección de anomalías.

2. OBJETIVOS

Objetivo general

Proponer un modelo matemático basado en datos usando aprendizaje profundo que pueda ser implementado en técnicas automáticas de tolerancia a fallos para la administración de un sistema HPC.

Objetivos específicos

- Analizar técnicas de monitoreo de estado de sistemas de cómputo de alto rendimiento para identificar estados críticos de fallos.
- Documentar modelos de aprendizaje profundo para el monitoreo de fallos en sistemas distribuidos que puedan ser implementadas en estrategias de tolerancia a fallos en arquitecturas computacionales de alto rendimiento (HPC).
- Establecer un modelo matemático basado en datos que permita construir técnicas de monitoreo de tolerancia a fallos basadas en aprendizaje profundo para arquitecturas HPC.
- Evaluar el modelo matemático propuesto para el monitoreo de tolerancia a fallas en un sistema distribuido con el fin de plantear un marco de trabajo para el soporte de actividades de administración del sistema HPC.

3. MARCO TEÓRICO

El marco teórico de este proyecto se divide en 2 grandes componentes que ayudan a brindar contexto, primero los antecedentes en relación a los fundamentos matemáticos aplicados en las técnicas de aprendizaje automático y aprendizaje profundo, así como el modelamiento de series temporales y luego conceptos en relación a la infraestructura de computación de alto rendimiento y sus problemas más relevantes.

3.1. Fundamentos matemáticos aplicados

El objetivo de esta sección es comprender las bases matemáticas sobre las cuales se fundamentan las técnicas de que son utilizadas para el desarrollo de presente proyecto de investigación.

3.1.1. Aprendizaje Automático El aprendizaje automático o también conocido como aprendizaje de máquinas (Machine Learning) ML, es el conjunto de algoritmos que son capaces de aprender a partir de los datos. Tom M. Mitchell en 1997, expresó: "Se dice que un programa de computadora aprende de una experiencia E respecto a unas clases de tareas T y con medida de desempeño P; si su desempeño en dichas tareas T, con una medida P, mejora con la experiencia E."[6]

- Una tarea T en ML es un problema que puede ser difícilmente abordado por programa escrito, una tarea puede ser descrita como un vector $x \in \mathbb{R}^n$ donde cada componente x_i del vector es una característica observada de un evento que se desea procesar.
- La experiencia E hace referencia a los datos con los cuales el sistema es alimentado y

[6] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. 2016, pág. 775.

entrenado, estos datos pueden estar etiquetados o no y de ello depende el tipo de tarea de aprendizaje que se aborde (aprendizaje supervisado o no supervisado).

- El desempeño P es una métrica que permite evaluar que tan bueno es el rendimiento de un modelo de ML.

El principal desafío en ML consiste asegurar que un algoritmo tenga un buen desempeño sobre nuevas observaciones o datos de entrada, este aspecto es conocido como la generalización. Una métrica comúnmente utilizada es el error de generalización (error de prueba o test) el cual es calculado usando nuevas observaciones, observaciones que no se usaron durante el entrenamiento. También a partir de los datos de entrenamiento se puede calcular una medida de error conocida como el error de entrenamiento. Los factores que determinan que tan bien un algoritmo de ML se desempeñará son la habilidad para:

- Hacer el error de entrenamiento pequeño.
- Reducir la brecha el error de entrenamiento y el de prueba.

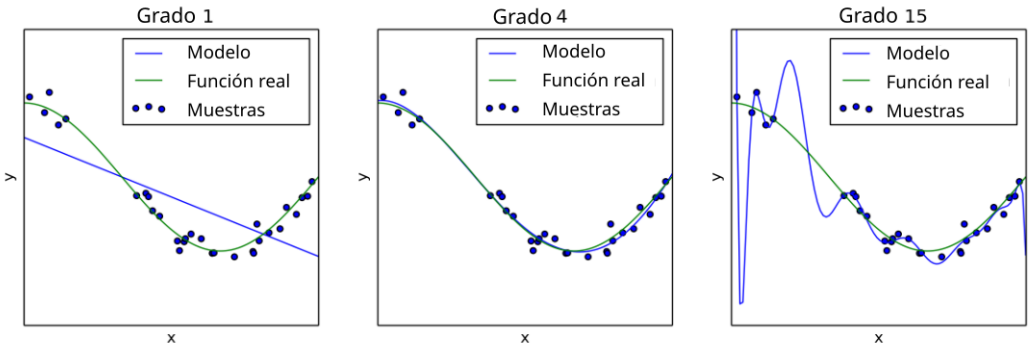


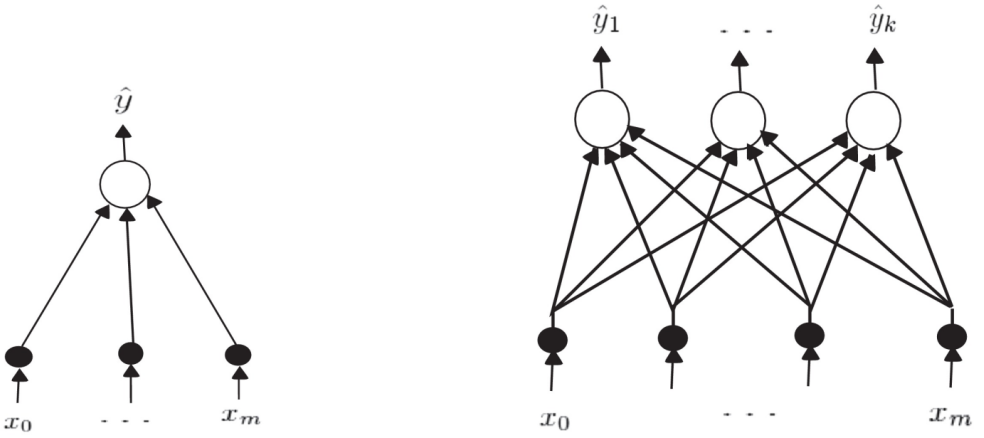
Figura 1. Ajuste de 3 modelos sobre un conjunto de datos de entrenamiento. (*izq*) Una función lineal se ajusta pero no logra capturar la curvatura, (*cen*) Una función de cuarto grado generaliza bastante bien, (*der*) Un polinomio de grado 15 se ajusta perfectamente a los datos pero esta no puede extraer la verdadera estructura.[7]

Estos dos factores corresponden a lo que se conoce sobregeneralización (underfitting) y sobre-ajuste (overfitting). El underfitting ocurre cuando el modelo no puede obtener un valor de error

suficientemente bajo en el conjunto de entrenamiento. El sobreajuste ocurre cuando la brecha entre el error de entrenamiento y el error de prueba es demasiado grande[6]. En la figura 1 se pueden observar estos dos factores.

3.1.2. Redes Neuronales Artificiales ANN Una red neuronal artificial (Artificial Neural Network) ANN es un conjunto de unidades de computo (neuronas) que dado un conjunto de entradas arroja un conjunto de salida. El modelo o arquitectura de ANN más básico es conocido como el perceptrón de una capa (figura 2). Existen varios tipos de perceptrones, uno de los más conocidos es el *linear threshold unit* (LTU) el cual recibe un vector de elementos de entrada x donde a cada elemento x_i se le asigna un peso w_i, luego se computa un escalar z el cual producto punto entre estos dos vectores $z = w_1x_1 + w_2x_2 + \cdots + w_nx_n = w^T \cdot x$ y se le aplica una función de umbral como la escalón heaviside(z)= $\tilde{y}$ donde:

$$\mathcal{H}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



(a) Perceptron unicapa con una salida

(b) Perceptron unicapa con *k* salidas

Figura 2. Representación gráfica de un perceptrón. [8]

Redes Neuronales Profundas Un perceptrón puede ser visto como una red neuronal de una capa, cuando se apilan varios perceptrones en secuencia se obtiene una ANN conocida como perceptron multicapa *Multi-Layer Perceptron* (MLP) Cuando una ANN tiene dos o más capas ocultas, aquellas que cuyas salidas son resultados intermedios no son relevantes, se le conoce como una red neuronal profunda[9].

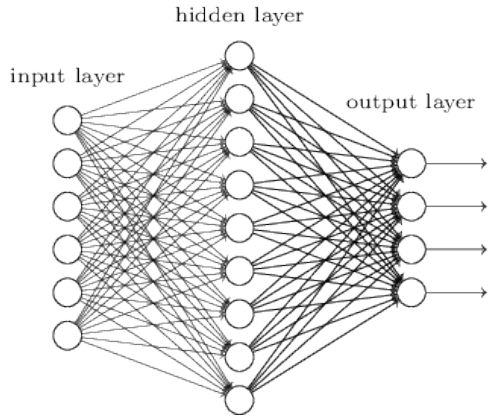


Figura 3. Red neuronal no-profunda

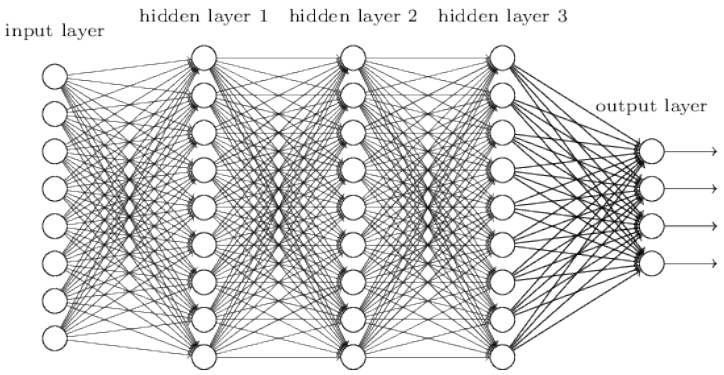


Figura 4. Red Neuronal Profunda

En las figuras 3 y 4 se dan ejemplos de representaciones de una red neuronal sencilla y una red profunda.

Backpropagation Cuando una red neuronal acepta una entrada x y produce una salida $\tilde{y}$, la información fluye "hacia adelante" a través de la red. Cuando las salidas son generadas, estas se pueden comprar con las salidas esperadas y calcular un error el cual será propagado de vuelta a través de las capas de la red, de ahí su nombre *backpropagation*. Esta técnica en los entornos académicos también es conocida como diferenciación automática acumulada hacia atrás, fue inventada por Linnainmaa (1970)[10] Esta técnica consiste en fijar una variable dependiente a ser

---

[9] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[10] Seppo Linnainmaa. «The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors». Tesis doct. Master's Thesis (in Finnish), Univ. Helsinki, 1970.

diferenciada y calcula su derivada con respecto a cada subexpresión recursivamente lo cual es equivalente a sustituir repetidamente la derivada de las funciones más externas usando la regla de la cadena:

$$\frac{\partial \tilde{y}}{\partial x} = \frac{\partial \tilde{y}}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial \tilde{y}}{\partial w_2}\frac{\partial w_2}{\partial w1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial \tilde{y}}{\partial w_3}\frac{\partial w_3}{\partial w2}\right)\frac{\partial w_2}{\partial w1}\right)\frac{\partial w_1}{\partial x} = \dots$$

Este proceso junto con la función de error permite recalcular los pesos $w_i^{(j)}$ donde el indice j indica una capa y el indice i indica la i-esima neurona de una capa.

3.1.3. Codificadores automáticos o *Autoencoders* Estos son una clase de redes neuronales que se componen de dos partes, un codificador (*encoder*) que comprime la información y un decodificador (*decoder*) que lleva a cabo una reconstrucción del vector de entrada original x en un vector de salida x', de manera que $x \approx x'$. (ver fig 5).

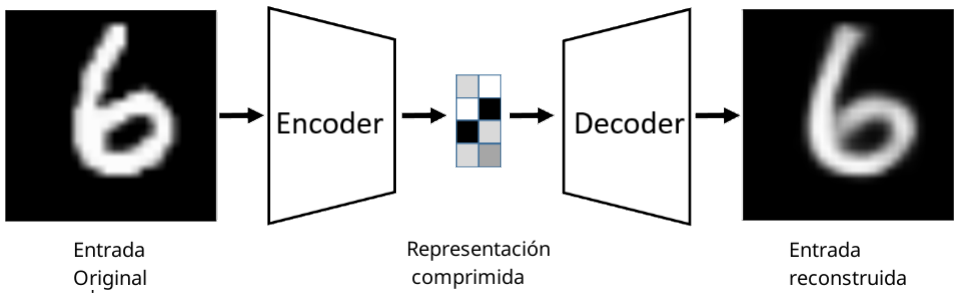


Figura 5. Ejemplo autoencoder.[11]

Esto no implica que, el *autoencoder* simplemente haga una copia del vector original, sino que internamente es capaz de aprender los patrones o representaciones de los datos de entrada, permitiendo que el vector comprimido z o vector del espacio latente pueda ser posteriormente reconstruido en un vector aproximadamente como el original.
En general, un autoencoder, se divide en 3 componentes principales:

Encoder: Sea x un vector de entrada, $f(x)$ funciona como una red neuronal profunda *DNN* clásica en la que a través de la activación de neuronas en capas con más pequeñas va emulando

una función de codificación para obtener un vector latente $z = f(x)$.

Espacio latente o cuello de botella: Usualmente z se conoce como el vector de representación latente. Esta capa intermedia es una capa oculta que describe en codificación especial la entrada, como $z = \sigma(Wx + b)$, aquí, σ es una función de activación de elementos como una función sigmoidea o una unidad lineal rectificada, W es una matriz de peso y b es un vector bias.

Decoder: Teniendo un $r = g(z)$ una función de decodificación para obtener una aproximación de x, como $x' = \sigma'(W'z + b')$.
Como resultado, un autoencoder es capaz de mapear una entrada a un espacio latente y luego reconstruir aquella entrada, pero internamente el espacio latente no necesariamente está estructurado, es decir, puntos similares en este espacio corresponden a entradas similares.

3.1.4. Autoencoder Variacional Un autoencoder variacional (VAE) es un modelo generativo probabilístico que se utiliza para aprender la distribución subyacente de un conjunto de datos[12]. Se basa en la premisa de que una muestra de datos x se genera a través de un proceso aleatorio que involucra una variable aleatoria desconocida z. La variable z se crea a partir de una distribución anterior específica $p_\theta(z)$, y el valor de los datos x se determina mediante la distribución condicional $p_\theta(x|z)$.
Un VAE se compone de dos componentes principales:

- **Modelo de reconocimiento:** Este modelo es un codificador probabilístico que genera, para una entrada específica x, una distribución sobre los posibles valores de z a partir de los cuales se generó x. Esta distribución se aproxima a la verdadera posterior, que es

[12] Dor Bank, Noam Koenigstein y Raja Giryes. *Autoencoders*. 2021. arXiv: `2003.05991 [cs.LG]`.

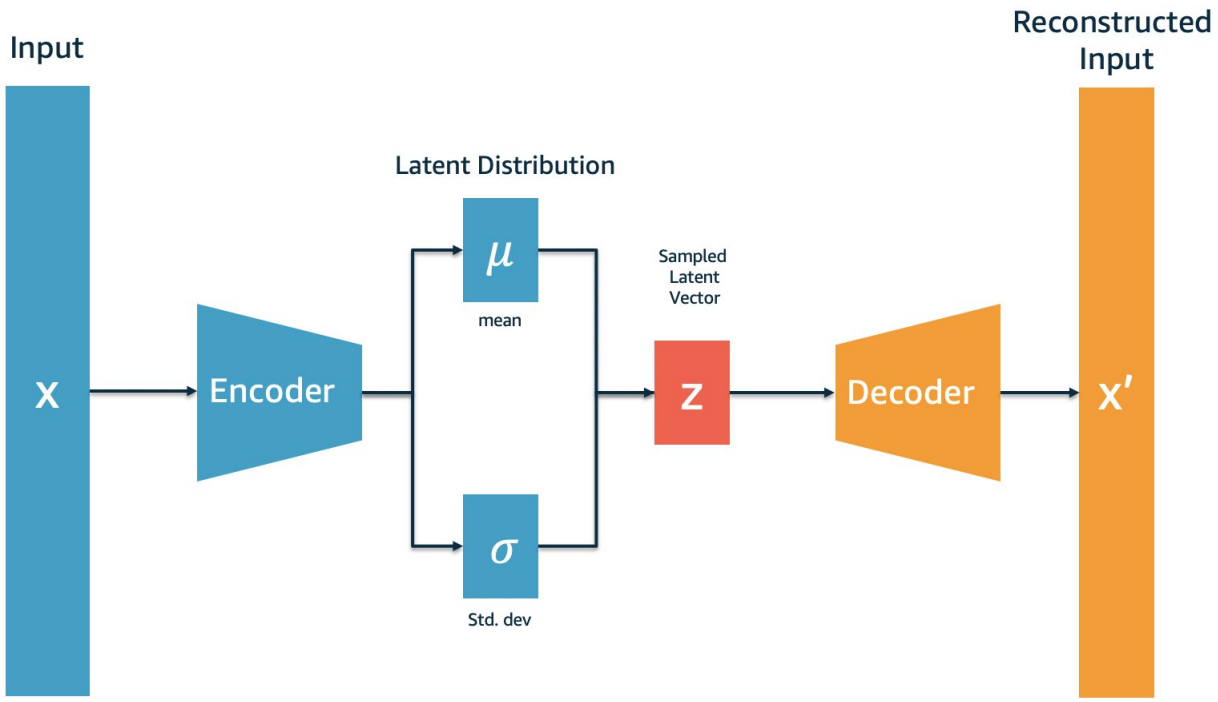
difícil de calcular directamente:

$$q_\phi(z|x) \approx p_\theta(z|x)$$

- **Modelo de decodificación:** Este modelo es un decodificador probabilístico que genera una distribución sobre los posibles valores de x para un z dado.

(Ver fig. 6).

Figura 6. Autoencoder variacional.



Fuente: Variational Autoencoder [imagen]. En: AWS Machine Learning Blog [Consultado el: 2 de julio de 2023]. Disponible en internet: url https://aws.amazon.com/blogs/machine-learning/deploying-variational-autoencoders-for-anomaly-detection-with-tensorflow-serving-on-amazon-sagemaker/

Entrenamiento de un VAE

Los parámetros del modelo, ϕ y θ, se aprenden de manera conjunta. El VAE se entrena para minimizar la Evidence Lower Bound (ELBO), también conocida como límite inferior variacional.

Expresión de la ELBO

La ELBO se define como la diferencia entre la entropía conjunta de x y z y la suma de la entropía marginal de x y la divergencia Kullback-Leibler (KL) entre $q_\phi(z|x)$ y $p_\theta(z|x)$:

$$L_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - D_{KL}(q_\phi(z|x)||p_\theta(z|x))$$

La primera parte de la expresión representa el error esperado de reconstrucción, mientras que la segunda parte cumple la función de regularizador.

La divergencia KL como regularizador

La divergencia KL mide la similitud entre dos distribuciones $p(x)$ y $q(x)$ y se define como:

$$D_{KL}[p(x)||q(x)] = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$$

En el contexto de los VAE, la divergencia KL mide la diferencia entre $q_\phi(z|x)$ y $p_\theta(z|x)$, que representa la similitud entre la aproximación y la verdadera posterior, siendo esta última desconocida e intratable. Este término es igual a cero si $q_\phi(z|x)$ coincide con la verdadera distribución posterior.

La divergencia KL se utiliza como regularizador en los VAE para evitar que el modelo se ajuste demasiado a los datos de entrenamiento. Esto se debe a que, si el modelo se ajusta demasiado, la distribución $q_\phi(z|x)$ se aproximará a la distribución empírica de los datos de entrenamiento, que es una distribución sesgada.

La estructura del espacio latente

El espacio latente de un VAE es un espacio de representación que captura la información importante de los datos de entrada. El hecho de que la divergencia KL se utilice como regularizador en los VAE tiene un impacto significativo en la estructura del espacio latente.

En general, los VAE con una divergencia KL grande tienden a generar espacios latentes más organizados. Esto se debe a que la divergencia KL penaliza las distribuciones $q_\phi(z|x)$ que son muy diferentes de las distribuciones $p_\theta(z|x)$.

La estructura del espacio latente tiene implicaciones importantes para las aplicaciones de los VAE. Por ejemplo, los VAE con espacios latentes organizados se pueden utilizar para generar nuevos datos que sean similares a los datos de entrenamiento.

El truco de reparametrización

Dado que z es una variable aleatoria y no podemos propagar gradientes directamente a través de ella, se introdujo lo que se conoce como el "truco de reparametrización". Este truco implica expresar la variable aleatoria $z \sim q_\phi(z|x)$ como una variable determinista $z = g_\phi(\varepsilon, x)$ utilizando otra variable aleatoria independiente ε. Para un caso de gaussiana univariante donde $z \sim p(z|x) = N(\mu, \sigma^2)$, la reparametrización resultaría en $z = \mu + \sigma\varepsilon$ con $\varepsilon \sim N(0, 1)$.

En un autoencoder variacional, el codificador probabilístico y el decodificador pueden representarse mediante una red neuronal, donde el codificador aproxima los parámetros de una distribución que generalmente es una gaussiana univariante con parámetros μ, σ. La elección de una distribución gaussiana se realiza porque se asume que la relación entre las variables en el espacio latente es considerablemente más simple que en el espacio de entrada original. En comparación con un autoencoder tradicional (AE), el VAE calcula una probabilidad de reconstrucción en lugar de un error de reconstrucción y proporciona una base teórica.

3.2. Conceptos relacionados con HPC

El objetivo de esta sección es comprender los conceptos relacionados al problema abordado en el desarrollo de presente proyecto de investigación.

3.2.1. Tolerancia a Fallos en HPC La tolerancia a fallos (FT) es una característica crucial que permite a los sistemas continuar operando a pesar de la presencia de fallos, ya sean de hardware o software. En infraestructuras HPC, los fallos son inherentes a la operación del sistema debido a la complejidad y escala de la plataforma, que incluye factores como la intrincada interconexión de componentes, cargas computacionales intensas, largos tiempos de operación, complejidad en los cálculos, entre otros. Dadas las particularidades en mención, hacen que estos sistemas HPC sean bastante susceptibles a que sus procesos de cómputo se vean interrumpidos, lo cual causa desperdicio de recursos como el tiempo, capacidad de cómputo y energía.
En este contexto, es crucial comprender que una falla puede ser tanto interna como externa al sistema bajo consideración. Los estados intermedios entre la aparición de una falla y la ocurrencia de un fallo se llaman errores. Las fallas pueden prevenirse, tolerarse, eliminarse o pronosticarse. A través de estas acciones, un sistema se vuelve 'resiliente' a las fallas [13].

3.2.2. Mecanismos de tolerancia a fallos Cuando un fallo en un sistema HPC ocurre, existen múltiples formas de lidiar con una eventualidad de ese tipo, aquellas técnicas son conocidas como mecanismos de tolerancia a fallos (Fault tolerance mechanisms).[14].
Los mecanísmos de tolerancia a fallos brindan la capacidad a un sistema de continuar operando

---

[13] Jared Herzog. «Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman». eng. En: *Software engineering notes* 40.1 (2015), págs. 51-52.

[14] Ifeanyi P. Egwutuoha et al. «A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems». En: *Journal of Supercomputing* 65.3 (2013), págs. 1302-1326. DOI: 10.1007/s11227-013-0884-0.

Figura 7. Tácticas de disponibilidad.

incluso si un fallo ocurre[15].

Detección de fallas En el ámbito de la tolerancia a fallos, la detección juega un papel fundamental en asegurar la integridad y confiabilidad de los sistemas informáticos. Dentro de esta categoría, diversas tácticas se despliegan de manera estratégica. La monitorización del sistema, el intercambio de mensajes ping/eco, el *hearbeat* y el uso de marcas de tiempo se destacan como herramientas esenciales para anticipar y detectar posibles fallas. Complementando estas estrategias, el monitoreo de condiciones, la comprobación de cordura y la votación ofrecen capas adicionales de seguridad. Por otro lado, la detección de excepciones y los autotest se centran en la identificación temprana y la validación del correcto funcionamiento del sistema. En conjunto,

---

[15] Michael Butler et al. *Proceedings of the Workshop on Methods, Models and Tools for Fault Tolerance (MeMToFT 2007).* Vol. 5454. Information Society Tecnologies, jul. de 2007. DOI: 10.1007/978-3-642-00867-2.

estos mecanismos forman un marco sólido para fortalecer la tolerancia a fallos, contribuyendo significativamente a la estabilidad y eficacia de los sistemas informáticos [13].

Recuperarse de fallas En el ámbito de la recuperación de fallas, las tácticas se subdividen en preparación y reparación, así como reintroducción. La preparación y reparación se basan en estrategias que incluyen desde repuestos redundantes hasta la capacidad de retroceder a un estado conocido ante la detección de fallas. El manejo de excepciones, actualizaciones de software, reintento, y la capacidad de ignorar comportamientos defectuosos complementan estas tácticas, brindando flexibilidad y adaptabilidad al sistema frente a fallas potenciales. La reintroducción, por otro lado, se materializa cuando un componente, una vez rehabilitado, es reintegrado al funcionamiento normal. Tácticas como operar en modo sombra, resincronización de estado, reinicio escalonado y reenvío continuo son empleadas con el objetivo de garantizar una reintegración efectiva y minimizar el impacto en los servicios durante el proceso de recuperación. En conjunto, estas estrategias forman un marco integral para abordar la recuperación de fallas, fortaleciendo la resiliencia y confiabilidad de los sistemas informáticos[13].

Prevenir fallas En lugar de simplemente detectar y recuperarse de fallas, la prevención de fallas se centra en evitar que ocurran en primer lugar. Estrategias como producir código de alta calidad mediante inspecciones, programación en pares y revisiones sólidas de requisitos son fundamentales. La táctica de retiro del servicio implica colocar temporalmente un componente fuera de servicio para mitigar posibles fallas antes de que afecten el sistema. Las transacciones, con su semántica transaccional, garantizan atomicidad, consistencia, aislamiento y durabilidad en sistemas de alta disponibilidad. La implementación de un modelo predictivo, combinado con un monitor, monitorea el estado del sistema y toma medidas correctivas antes de que alcance umbrales críticos. La prevención de excepciones involucra técnicas como clases de excepción, código de corrección de errores y el uso de punteros inteligentes y contenedores para evitar excepciones en el sistema. Incrementar el conjunto de competencias de un programa implica diseñarlo para manejar más casos de manera proactiva, evitando que un componente se encuentre

fuera de su conjunto de competencias y contribuyendo así a la prevención efectiva de fallas [13].

3.2.3. Caracterización de fallos Si bien los mecanismos de tolerancia a fallos son esenciales para mantener la operación continua de sistemas HPC, es importante reconocer que, en última instancia, la falla es inevitable. Los sistemas, debido a su complejidad y escala, están expuestos a una variedad de fallos que van más allá de lo previsible.

La caracterización de fallos se vuelve desafiante dada la diversidad de factores que pueden contribuir a su ocurrencia. La diversidad de fallos, especialmente los de hardware, se clasifica en Detectable Correctable Errors (DCE), Detectable Uncorrectable Errors (DUE) y Silent Errors (SE) o Silent Data Corruption (SDC)[16]. Los DCEs, gestionados por mecanismos de hardware, son errores corregibles pero sutiles. Los DUEs interrumpen la ejecución, mientras que los SDCs corrompen datos de manera silenciosa.

3.3. Métricas de validación

El rendimiento de los modelos de aprendizaje profundo puede ser evaluado a través de diferentes metricas en función del tipo de tarea que va a desempeñar, en este caso, se evaluará la capacidad de un autoencoder variacional para detectar anomalías, es decir, clasificar su salida como una variable binaria, verdadera o falsa, las metricas empleadas en estre trabajo son las siguientes:

3.3.1. Precisión También llamada confianza indica la proporción de casos predichos como positivos que son realmente positivos[17]. Es decir, mide la precisión de los casos positivos predichos en comparación con el número total de casos predichos como positivos. La precisión es especialmente relevante en campos como el aprendizaje automático, la minería de datos y

---

[16] Nuria Losada. «Application-level Fault Tolerance and Resilience in HPC Applications». Tesis doct. Universidade da Coruña, 2018, pág. 145.

[17] Peter Flach y Meelis Kull. «Precision-Recall-Gain Curves: PR Analysis Done Right». En: *Advances in Neural Information Processing Systems*. Ed. por C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015.

la recuperación de información. En términos de la tasa de precisión verdadera (tpa), se define como el número de verdaderos positivos (TP) dividido por el número total de casos predichos como positivos (PP), según la ecuación:

$$Precision = Confidence = tpa = \frac{TP}{PP} = \frac{A}{(A+B)}$$

3.3.2. Recall También conocida como Sensibilidad o Tasa de Verdaderos Positivos se refiere a la proporción de casos positivos reales que son correctamente identificados como positivos por el modelo predictivo[17]. Es decir, mide la cobertura de los casos positivos reales por la regla de predicción positiva. En términos de la tasa verdaderamente positiva (tpr) en el análisis de la curva ROC, el recall se define como el número de verdaderos positivos (TP) dividido por el número total de casos positivos reales (RP), según la ecuación:

$$Recall = Sensitivity = tpr = \frac{TP}{RP} = \frac{A}{(A+C)}$$

3.3.3. F1 score Es una medida de la precisión de un modelo en un conjunto de datos. Se utiliza para evaluar sistemas de clasificación binaria, que clasifican ejemplos en positivos o negativos. El F-score es una medida que combina la precisión y el recall en una sola métrica, proporcionando una evaluación global del rendimiento del modelo en tareas de clasificación[18]. El F-score, o F-measure, se calcula utilizando la siguiente fórmula:

$$F = \frac{2 \cdot P \cdot R}{P + R}$$

Donde:

- P es la precisión, calculada como $P = \frac{TP}{TP+FP}$, siendo TP el número de verdaderos positivos

[18] David J. Hand, Peter Christen y Nishadi Kirielle. «F*: an interpretable transformation of the F-measure». En: *Machine Learning* 110.3 (2021), págs. 451-456. DOI: 10.1007/s10994-021-05964-1.

y FP el número de falsos positivos.

- R es el recall, calculado como $R = \frac{TP}{TP+FN}$, siendo TP el número de verdaderos positivos y FN el número de falsos negativos.

El marco teórico proporciona los fundamentos matemáticos y las teoría esencial para el desarrollo de los modelos de aprendizaje automático y profundo, así como una comprensión de los desafíos y técnicas avanzadas como las redes neuronales y los autoencoders variacionales, además que describe las métricas esenciales para evaluar la capacidad de un modelo para detectar anomalías de manera efectiva y precisa. A continuación, en el capitulo 4 se presenta el estado del arte.

4. ESTADO DEL ARTE

Teniendo en cuenta una serie de conceptos presentados en el marco teórico, a continuación una serie de antecedentes son expuestos en este capítulo, en esta revisión del estado del arte se abordan enfoques relacionados con los nuevos avances relacionados con el problema de investigación.

Existen diferentes técnicas de software para mitigar y evitar la influencia de la Silent Data Corruption (SDC), como por ejemplo, extensiones de lenguajes de programación como Rolex[19], también implementación de librerías como CRAFT[20] que soportan la implementación de protocolos Checkpoint/Restart, así también algoritmos como por ejemplo los "gossips-inspired"presentados en Casas (2019)[21].

En Benoit (2017)[22], algunas de las estrategias de checkpoint son revisadas y un modelo de rendimiento es presentado. De acuerdo con Benoit (2018)[23], hacer checkpoints es un estándar de-facto cuando se abordan errores tipo fail-stop.

---

[19] Dewan Ibtesham, Kurt B. Ferreira y Dorian Arnold. «A checkpoint compression study for high-performance computing systems». En: *International Journal of High Performance Computing Applications* 29.4 (2015), págs. 387-402. DOI: 10.1177/1094342015570921.

[20] Faisal Shahzad et al. «CRAFT: A library for easier application-level Checkpoint/Restart and Automatic Fault Tolerance». En: *IEEE Transactions on Parallel and Distributed Systems* 30.3 (2018), págs. 501-514. DOI: 10.1109/TPDS.2018.2866794. arXiv: 1708.02030.

[21] Marc Casas, Wilfried N. Gansterer y Elias Wimmer. «Resilient gossip-inspired all-reduce algorithms for high-performance computing: Potential, limitations, and open questions». En: *International Journal of High Performance Computing Applications* 33.2 (2019), págs. 366-383. DOI: 10.1177/1094342018762531.

[22] Anne Benoit, Saurabh K. Raina e Yves Robert. «Efficient checkpoint/verification patterns». En: *International Journal of High Performance Computing Applications* 31.1 (2017), págs. 52-65. DOI: 10.1177/1094342015594531.

[23] Anne Benoit et al. «Multi-level checkpointing and silent error detection for linear workflows». En: *Journal of Computational Science* 28 (2018), págs. 398-415. DOI: 10.1016/j.jocs.2017.03.024.

En Bosilca (2018)[24] un protocolo de detección de fallos (FD) es presentado como prerrequisito para la mitigación de fallos y estableciendo que el protocolo FD es un componente clave para cualquier infraestructura que requiera resiliencia.

De acuerdo con Lima (2021)[25], el mantenimiento predictivo es una solución prometedora porque se encarga de estimar el mejor momento para intervenir un sistema antes de que un fallo ocurra, por tanto evitando que un sistema falle y sucedan detenciones no planeadas.

En recientes años, nuevos trabajos han sido desarrollados, por ejemplo, en Guo (2021)[26] una de las nuevas soluciones propuestas se llama PARIS, el cual es una aplicación de resiliencia basada en simulaciones de inyección de fallos.

En Li (2021)[27] una visualización llamada SpotSDC es presentada; este sistema facilita el análisis de la resiliencia de un programa a la SDC.

Respecto al uso de técnicas de aprendizaje profundo en el campo de la computación de alto rendimiento, en Li (2022)[28] técnicas de Deep Reinforcement Learning (DRL) son utilizados en conjunto a las redes neuronales generativas (GAN) para la optimización de la calendarización de tareas.

---

[24] George Bosilca et al. «A failure detector for HPC platforms». En: *International Journal of High Performance Computing Applications* 32.1 (2018), págs. 139-158. DOI: 10.1177/1094342017711505.

[25] André Luis da Cunha Dantas Lima et al. «Smart predictive maintenance for high-performance computing systems: a literature review». En: *Journal of Supercomputing* 77.11 (2021), págs. 13494-13513. DOI: 10.1007/s11227-021-03811-7.

[26] Luanzheng Guo, Dong Li e Ignacio Laguna. «PARIS: Predicting application resilience using machine learning». En: *Journal of Parallel and Distributed Computing* 152 (2021), págs. 111-124. DOI: 10.1016/j.jpdc.2021.02.015. arXiv: 1812.02944.

[27] Zhimin Li et al. «SpotSDC: Revealing the Silent Data Corruption Propagation in High-Performance Computing Systems». En: *IEEE Transactions on Visualization and Computer Graphics* 27.10 (2021), págs. 3938-3952. DOI: 10.1109/TVCG.2020.2994954.

[28] Jingbo Li et al. «GARLSched: Generative adversarial deep reinforcement learning task scheduling optimization for large-scale high performance computing systems». En: *Future Generation Computer Systems* 135 (2022), págs. 259-269. DOI: 10.1016/j.future.2022.04.032.

También en Yang (2022)[29] las técnicas de DRL son utilizadas para fortalecer y optimizar un algoritmo voraz para tareas de asignación de recursos en entornos HPC en la nube.

Respecto a las técnicas de detección de anomalías, en He (2016)[30] se llevan pruebas utilizando varias técnicas automatizadas con el objetivo de reducir el esfuerzo manual en la detección de anomalías sobre logs, en este estudio se utilizan 6 métodos, 3 supervisados (Regresión logística, Árboles de decisión y máquinas de soporte vectorial) y 3 no-supervisados (Clustering, Análisis de Componentes Principales *PCA* y Minería de Invarianzas), en donde los métodos supervisados resultaron en rendimientos similares entre sí y los métodos no-supervisados la minería de invarianzas presentó un rendimiento ligeramente superior.

En Kwon(2019)[31] se lleva a cabo una inspección de varias técnicas para la detección de anomalías, en este caso sobre datos de redes para analizar el tráfico a través de ellas. Allí son presentadas diferentes técnicas entre las que destacan las técnicas de apredizaje profundo.

En Farzad (2020)[32] se propone un modelo no-supervisado para la detección de anomalías sobre logs, el modelo presentado corresponde a un ensemble para llevar a cabo dos procesos, primero, se usa un autoencoder para realizar una extracción de características que alimentan un bosque asilado *Isolation Forest*, luego se toman las características positivas que permiten reconstruir logs para alimentar un segundo autoencoder permitirá establecer un umbral *threshold* para llevar a cabo la detección de anomalías sobre los logs.

---

[29] Yuanhao Yang y Hong Shen. «Deep Reinforcement Learning Enhanced Greedy Optimization for Online Scheduling of Batched Tasks in Cloud HPC Systems». En: *IEEE Transactions on Parallel and Distributed Systems* 33.11 (2022), págs. 3003-3014. DOI: 10.1109/TPDS.2021.3138459.

[30] Shilin He et al. «Experience Report : System Log Analysis for Anomaly Detection». En: *2016 IEEE 27th International Symposium on Software Reliability Engineering*. 2016. DOI: 10.1109/ISSRE.2016.21.

[31] Donghwoon Kwon et al. «A survey of deep learning-based network anomaly detection». En: *Cluster Computing* 22.s1 (2019), págs. 949-961. DOI: 10.1007/s10586-017-1117-8.

[32] Amir Farzad y T Aaron Gulliver. «Unsupervised log message anomaly detection». En: *ICT Express* 6.3 (2020), págs. 229-237. DOI: 10.1016/j.icte.2020.06.003.

Al igual que en Kwon (2019)[31], la inspección realizada por Yadav (2020)[33] son revisadas diferentes técnicas de aprendizaje profundo, en esta revisión es mencionado el uso principalmente de ténicas Redes Neuronales Recurrentes LSTM, así como GRU, para el caso particular de logs provenientes de systemas HPC.

En el paper de Sinha (2022)[34] se presenta un enfoque usando Deep Learning que evalúa las capacidades de una red neuronal convolucional *CNN* para encontrar patrones que permitan detectar anomalías en logs, para esto, el modelo aprende las variaciones que existen entre entre una ejecución normal y una ejecución que representa un comportamiento anormal a través de la construcción de imágenes.

En Le (2022)[35] se lleva a cabo una comparativa entre 5 modelos de Deep Learning sobre 4 datasets, donde los modelos presentaron rendimientos similares con una ligera superioridad en aquellos basados en LSTMs, ninguno de los modelos evaluados se basó en AutoEncoders.

El reporte llevado acabo por Chen (2022)[36] es una actualización del realizado en He (2016)[30] con la diferencia en que en Chen (2022) son revisadas técnicas basadas en Deep Learning

El trabajo más reciente, Landauer (2023)[37] corresponde a una inspección en donde se reporta el uso de técnicas de Deep Learning a través de técnicas de bibliometría para evaluar la popularidad

---

[33] Rakesh Bahadur Yadav. «A Survey on Log Anomaly Detection using Deep Learning». En: (2020), págs. 1215-1220.

[34] Rohit Sinha et al. «Anomaly Detection Using System Logs: A Deep Learning Approach». En: *International Journal of Information Security and Privacy* 16.1 (), págs. 1-15. DOI: 10.4018/IJISP.285584.

[35] Van-hoang Le y Hongyu Zhang. «Log-based Anomaly Detection with Deep Learning : How Far Are We ?» En: *44th International Conference on Software Engineering (ICSE '22), May 21â•fi29, 2022, Pittsburgh, PA, USA*. Association for Computing Machinery, 2022. DOI: 10.1145/3510003.3510155. arXiv: arXiv:2202.04301v2.

[36] Zhuangbin Chen y Michael R Lyu. *Experience Report : Deep Learning-based System Log Analysis for Anomaly Detection*. Inf. téc. 2022. DOI: https://doi.org/10.1145/1122445.1122456. arXiv: arXiv:2107.05908v2.

[37] Max Landauer et al. «Deep learning for anomaly detection in log data: A survey». En: *Machine Learning with Applications* 12 (2023), pág. 100470. DOI: https://doi.org/10.1016/j.mlwa.2023.100470.

de diversos métodos utilizados para la detección de anomalías, esta publicación corresponde una inspección de 62 diferentes papers que presentan múltiples enfoques, entre los cuales aparece el uso de AutoEncoders para abordar la detección de anomalías en logs.

Finalmente como conclusión de la revisión del estado del arte se encuentran diferentes formas de abordar el reto de garantizar la tolerancia a fallos de un sistema HPC, en donde el análisis de logs representa una alternativa interesante a través del entrenamiento de un modelo de aprendizaje profundo que permita detectar anomalías que pueden representar potenciales fallos en la operación del sistema.

En los siguientes capítulos se hablará de la metodología utilizada para llevar a cabo el proyecto5, el trabajo de modelado usado para este proyecto6, seguido de la evaluación de los resultados7 y las conclusiones8.

5. METODOLOGÍA

Al ser este un proyecto de matemática aplicada sobre una área indisciplinar de la ingeniería de sistemas, como lo es la ciencia de datos, se aprovechará uno de los modelos estándar más populares y utilizados dentro de la industria para el desarrollo de proyectos de minería de datos y la construcción de software basado en modelos matemáticos sobre grandes volúmenes de datos. Este modelo es conocido como CRISP-DM por sus siglas en inglés *Cross Industry Standard Process for Data Mining*[38], ver figura 8, este servirá como base para el desarrollo de este proyecto de investigación.

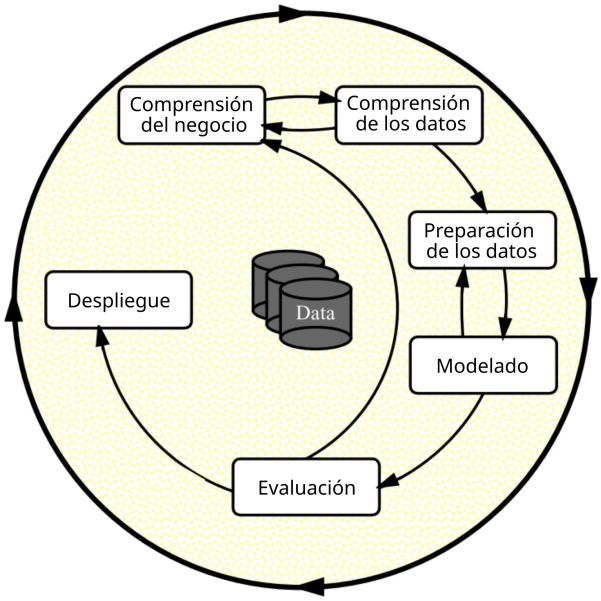


Figura 8. Fases del modelo para la minería de datos CRISP-DM.[39]

CRISP-DM divide el proceso de minería de datos en 6 fases, los cuales se basan en la compren-

[38] Rüdiger Wirth. «CRISP-DM : Towards a Standard Process Model for Data Mining». En: *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining* 24959 (1995), págs. 29-39. DOI: 10.1.1.198.5133.

sión del problema (reglas del negocio), comprensión de los datos, preparación de los datos, el modelado, la evaluación del modelo y el despliegue del modelo implementado[40].

Al ser este un proyecto de investigación, la fase final de despliegue y puesta en producción no será abordada. Teniendo en cuenta que el principal foco del proyecto es el modelado, para esta fase en particular me apoyaré en la metodología SEMMA (*Sample, Explore, Modify, Model and Assess*) desarrollada por el instituto SAS [41] [42] [43]. Descrito en la figura 9.

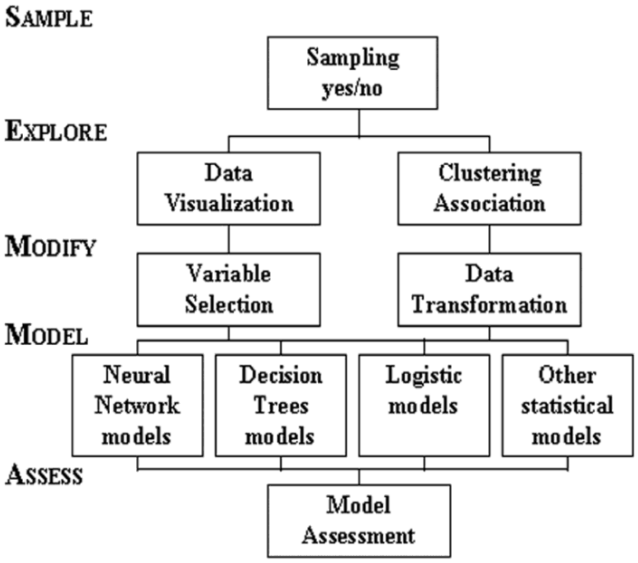


Figura 9. Diagrama SEMMA.[44]

Como sus siglas en inglés lo indican, este proceso se divide en 5 pasos iterativos, los cuales son:

- *Sample*: Esta fase se enfoca en la selección de una muestra lo suficientemente grade para

---

[40] Óscar Marbán, Gonzalo Mariscal y Javier Segovia. *A Data Mining & Knowledge Discovery Process Model*. Ed. por Julio Ponce y Adem Karahoca. February. I-Tech, 2009, pág. 436. DOI: 10.5772/6438. arXiv: 1411.2705.

[41] Randall Matignon. *Neural Network Modeling Using Sas Enterprise Miner*. Authorhouse, 2005.

[42] SAS Institute Inc. *SAS® Enterprise Miner™ 15.1: Reference Help*. URL: https://documentation.sas.com/?docsetId=emref&docsetTarget=n061bzurmej4j3n1jnj8bbjjm1a2.htm&docsetVersion=15.1&locale=en (visitado 17-09-2019).

[43] Dursun Delen David L. Olson. *Advanced Data Mining Techniques*. 1st ed. Springer, 2008.

realizar hallazgos relevantes, pero lo suficientemente pequeña para que el modelo corra ágilmente.

- *Explore*: Los datos son explorados gráfica y estadísticamente para familiarizarse con la naturaleza del dataset.
- *Modify*: Acá los datos son limpiados, acomodados y transformados. Se toman decisiones respecto a datos atípicos (outliers) y valores nulos o corruptos.
- *Model*: A partir de los datos preprocesados y normalizados se procede a construir, desplegar o implementar un modelo predictivo.
- *Assess*: Finalmente, antes de reiniciar el proceso, se validan los resultados en base a las métricas elegidas.

A partir de la metodología presentada se lleva a cabo el desarrollo de en el capítulo 6 de modelado en donde se aborda según CRISP-DM la comprensión de los datos utilizados en el proyecto, el modelado propuesto para el tratamiento de los datos y finalmente en el capítulo 7 la evaluación del rendimiento del modelo.

6. MODELADO

6.1. Datos de entrada

Para el presente proyecto se contó como insumo principal los datos provenientes de la plataforma HPC del laboratorio de SuperComputación y Cálculo Científico SC3UIS de la Universidad Industrial de Santander, y del Laboratorio Nacional de Computación de Alto Rendimiento (NLHPC) de Chile, los cuales corresponden a los logs de la herramienta SLURM generados como resultado de la operación del sistema.

SLURM (Simple Linux Utility for Resource Management) [45] es una herramienta de código abierto que facilita la gestión de recursos en plataformas HPC a través de la gestión de recursos, planificación de trabajos y el monitoreo de los trabajos en ejecución. Entre los componentes principales de SLURM está slurmctld el cual es el servicio o demonio que centraliza la gestión del estado de la plataforma HPC, programa trabajos y distribuye recursos. Inicialmente en las fase se preprocesamiento se usaron los datos del SC3UIS. Para el entrenamiento de este modelo, se utilizó como insumo los logs generados por 4 días de operación del cluster del NLHPC, los cuales suman en total aproximadamente 2.5 millones de registros.

6.1.1. Pre-procesamiento La fase de preprocesamiento consiste en transformar los datos para que puedan ser usados por los modelos de aprendizaje profundo, los cuales operan con valores numéricos, para esto primero es necesario estructurarlos, este proceso comúnmente se le conoce como *parsing*. Los datos en su estado original provienen en archivos de texto plano, al realizar una inspección visual de datos en su estado original, se puede observar que cada línea está compuesta por fragmentos que varían y fragmentos que son constantes, el primer fragmento

---

[45] SLURM. *Simple Linux Utility for Resource Management*. URL: `https://slurm.schedmd.com/overview.html` (visitado 14-02-2024).

de cada línea es común y representa un timestamp correspondiente al momento en el que fue registrado lo que a partir de ahora denominaremos como un 'evento del sistema' o simplemente 'evento', como por ejemplo se puede ser ilustrar mediante el siguiente fragmento de cinco líneas:

```
[2023-10-11T03:22:02.735] _job_complete: JobId=26722067_26871(26773803) WEXITSTATUS 0
[2023-10-11T03:22:02.737] _job_complete: JobId=26722067_26871(26773803) done
[2023-10-11T03:22:02.737] reconfigure_slurm: completed usec=325615
[2023-10-11T03:22:02.923] _job_complete: JobId=26722067_26874(26773806) WEXITSTATUS 0
[2023-10-11T03:22:02.925] _job_complete: JobId=26722067_26874(26773806) done
```

Cada evento registrado brinda un descripción contenida en los fragmentos constantes que acompañan los fragmentos variables, como por ejemplo se pueden encontrar palabras clave como JobId, Node, Partition, entre otras las cuales brindan información más detallada y permiten diferenciar por ejemplo tareas o flujos de trabajo.

Para llevar a cabo el proceso de *parsing* en mención se optó entonces por construir expresiones regulares que permitieran discriminar varios tipos de eventos como por ejemplo:

```
#Ejemplo de un diccionario con 3 patrones de eventos.
patterns = {
    "Job Complete Status": r"\[(\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d{3})\]
        _job_complete: JobId=(\d+(?:_\d+\(\d+\))?) WEXITSTATUS (\d+)",
    "Job Complete":      r"\[(\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d{3})\]
        _job_complete: JobId=(\d+(?:_\d+\(\d+\))?) done",
    "reconfigure_slurm": r"\[(\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d{3})\]
        reconfigure_slurm: (.*)"
    }
```

Cada llave corresponde al nombre que se le asignó al evento y el valor corresponde al registro completo crudo el cual sigue ciertos patrones identificados a través de la observación del mensaje, para ajustar los valores se utilizaron expresiones regulares que permiten construir

grupos de patrones, como por ejemplo, en el primer patrón, *Job Complete*, la expresión regular: `"\[(\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d{3})\]"` captura la fecha porque toma primero el corchete '[' luego toma 4 dígitos del año con '4', el guión '-', 2 dígitos del mes '2', etcétera, luego de la fecha, se toma la cadena de caracteres `"_job_complete: JobId="`, luego hay una expresión `"(\d+(?:_\d+\(\d+\))?)"` que captura el JobId y finalmente con la expresión `" done"` para concluir.

6.1.2. Extracción y representación de Características Una vez construido un diccionario de datos, se pueden agrupar los registros que estén asociados a los identificadores de las tareas en ejecución o JobId para generar secuencias de eventos, por ejemplo para un determinado JobId se puede tener la siguiente secuencia y conteo de eventos respectivamente:

```
['Job Submission', 'Update Job Setting', 'Update Job Slurm', 'Job Started',
'Job Complete Status', 'Job Complete']
```

```
Counter({'Job Submission': 1, 'Update Job Setting': 1, 'Update Job Slurm': 1,
'Job Started': 1, 'Job Complete Status': 1, 'Job Complete': 1})
```

De esta manera se pueden construir dos matrices en donde para la primera matriz cada fila corresponda a la secuencia de eventos dado un JobId y para la segunda cada fila corresponda al número de veces que apareció el evento. Para simplificar la secuenciación de eventos se optó por solo registrar la primera vez que aparece un tipo de evento en la secuencia, de manera que para el ejemplo anterior las dos listas del anterior ejemplo se convertirían en los vectores:

$$\langle 1, 2, 3, 4, 5, 6 \rangle$$

$$\langle 1, 1, 1, 1, 1, 1 \rangle$$

De esta manera los logs que se recibieron inicialmente quedaron representados numéricamente para entrenar el modelo.

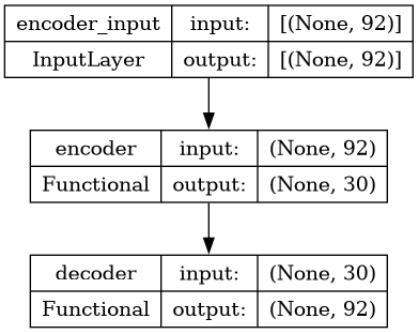


Figura 10. VAE

6.2. Arquitectura utilizada

Los Autoencoders Variacionales (VAE) son modelos cuyo propósito es el de aprender una distribución subyacente en un espacio latente que corresponde a las representaciones de los datos con los que fue entrenado, esto es útil para llevar a cabo detección de anomalías porque los datos anómalos se van a desviar significativamente de la distribución aprendida por el modelo.
El modelo de reconocimiento o encoder posee una capa de entrada, seguida de una capa de perceptrones densamente conectada cuyo tamaño es la mitad de los vectores x de entrada, que alimenta a su vez dos capas, una que se encargan de aprender las representaciones de la media y las de la desviación estándar, el tamaño de estas capas será $1/3$ de los vectores x de entrada. La siguiente capa aprovechando el truco de la reparametrización, genera una muestra a partir de las salidas de las dos capas anteriores y finalmente arrojar como salida un vector z cuyo tamaño es $1/3$ de la capa de entrada.
El modelo de decodificación recibe como entrada un vectores z cuyo tamaño es $1/3$ de los vectores x, al ser simétrico, el tamaño de la capa intermedia corresponde a $1/2$ del tamaño de los x y finalmente la capa de salida genera vectores del tamaño de los vectores x originales.

6.2.1. Implementación de arquitectura *Autoencoder* Para la implementación del modelo, se utilizó el lenguaje de programación Python y el framework Tensorflow con el backend de Keras, para el entrenamiento se construyó un dataset con los logs de slurm de los días 12, 13,

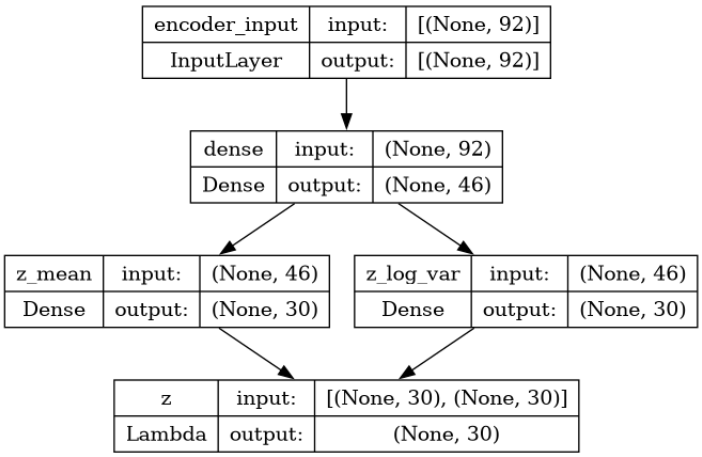


Figura 11. Encoder

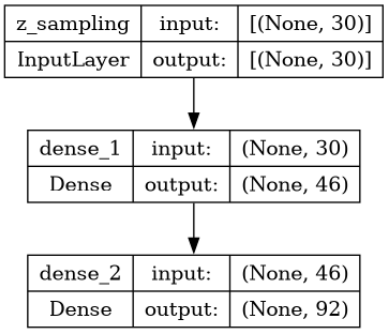


Figura 12. Decoder

14 y 15 de octubre del 2023 los cuales contenían 2.439.658 líneas, los cuales contenían 41.819 JobId diferentes, que luego del procesamiento se convirtieron en 41.819 secuencias de eventos, los eventos caracterizados fueron 46. De esas 41.819 secuencias, 2.462 tenían eventos identificados como anómalos o errores, estas secuencias no fueron tenidas en cuenta para el entrenamiento, pero si fueron usadas posteriormente para evaluar el rendimiento del modelo.

6.3. Entrenamiento del modelo

El conjunto de datos, como en cualquier modelo de aprendizaje automático *(Machine learning)*, ha sido dividido en dos subconjuntos, uno de entrenamiento y otro de prueba, el conjunto de entrenamiento se construyó con 36.799 secuencias normales y el de pruebas con 5.021 registros, de los cuales 2.462 son anómalos y 2.559 son normales.

Para las capas intermedias tanto del encoder como del decoder utilizaron funciones de activación ReLu, y la salida del decoder utilizó como activación una sigmoide, el optimizador utilizado fue Adam con una taza de aprendizaje de 0,0001.

Respecto a los recursos de infraestructura utilizados en el entrenamiento, se utilizó un nodo del cluster SC3UIS cuyo nombre clave es Yaje, este es un nodo HPE ProLiant ML350 con un procesador de 9na gen Intel(R) Xeon(R) CPU E5-2609 v3 @ 1.90GHz – 6 Cores, 48GB RAM y una GPU NVIDIA GeForce GTX Titan X 12 GB.

Una vez abordado el modelamiento en el presente capítulo para la detección de anomalías sobre los datos dispuestos utilizando el VAE se tiene comprensión de la naturaleza de los datos, cómo fueron preprocesados, cómo se construyeron las características, además, también se presentó la arquitectura del VAE y finalmente se habló del entrenamiento. En el siguiente capítulo 7 se lleva a cabo la evaluación del rendimiento del VAE en la detección de anomalías, utilizando métricas como precisión, recall y F1 score. Además se presentan los resultados obtenidos y se discute la efectividad del modelo para identificar secuencias anómalas y normales, validando su aplicación para datos de plataformas HPC.

7. EVALUACIÓN Y RESULTADOS

En este capítulo el enfoque propuesto en el anterior será evaluado. Para esto se utilizarán las métricas descritas en el marco teórico, como lo son la precisión, sensibilidad y F1-score para evaluar la habilidad del modelo en la detección de anomalías. A medida que se va entrenando el modelo se puede observar que el función de pérdida (ELBO) se va minimizando con el paso de los epochs.

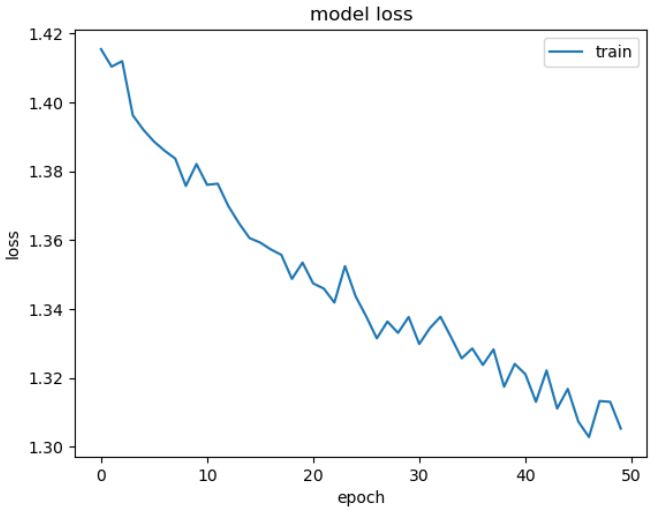


Figura 13. Minimización de función de pérdida ELBO para los primeros 50 epochs.

Esta disminución indica que el modelo está mejorando su capacidad para representar y reconstruir los datos de entrada. A medida que la pérdida disminuye, el modelo se ajusta mejor a los datos que en el espacio latente representan la distribución normal de las secuencias que corresponden a una operación del sistema sin fallas, lo que sugiere una mayor precisión en la detección de anomalías. La tendencia descendente es un signo positivo de que el modelo está convergiendo hacia una solución óptima.

7.1. Reportes

A continuación se presenta una tabla que resume los resultados para el modelo entrenado utilizando como parametro: 200 epoch, 250 tamaño de lotes, 1000 pasos por epoch, optimizador adam con una taza de aprendizaje de 0,0001.

	Precisión	Sensibilidad	F1-Score
Normal	1.00	0.89	0.94
Anomalía	0.90	1.00	0.95
Promedio	0.95	0.94	0.94

Tabla 1. Métricas reportadas

Los resultados presentados en la tabla 1 muestran un desempeño sobresaliente del modelo, evidenciado por su alto nivel en cada una de las medidas consideradas. Según estos resultados, aproximadamente el 94 % de los casos se clasifican correctamente.

Con el objetivo de visualizar los datos clasificados en el conjunto de prueba, se llevó a cabo un análisis de componentes principales (PCA) a la transformación de estos en un espacio latente (es decir aplicando la red encoder). Los dos primeros componentes se muestran en las figuras 15 y 14, junto con las etiquetas predichas y reales, respectivamente.

Estas representaciones gráficas revelan una similitud entre las etiquetas predichas y las reales, lo cual ayuda a explicar las altas medidas de desempeño presentadas en la Tabla 1. En la figura 14, se observan tres grupos distintivos, donde dos de ellos corresponden claramente a errores (naranja), mientras que el tercer grupo representa procesos normales (azul). Sin embargo, se nota una superposición entre el grupo de procesos normales y uno de errores, lo que sugiere que la identificación de errores es una tarea compleja debido a la falta de una diferenciación clara incluso en un espacio latente.

La visualización mediante PCA es un apoyo para complementar las métricas presentadas, ya que proporcionan una representación un poco más intuitiva del rendimiento del modelo en comparación a las métricas numéricas que lo hacen de forma cuantitativa.

Se optó también por aumentar el número de epochs para minimizar aún más la función de

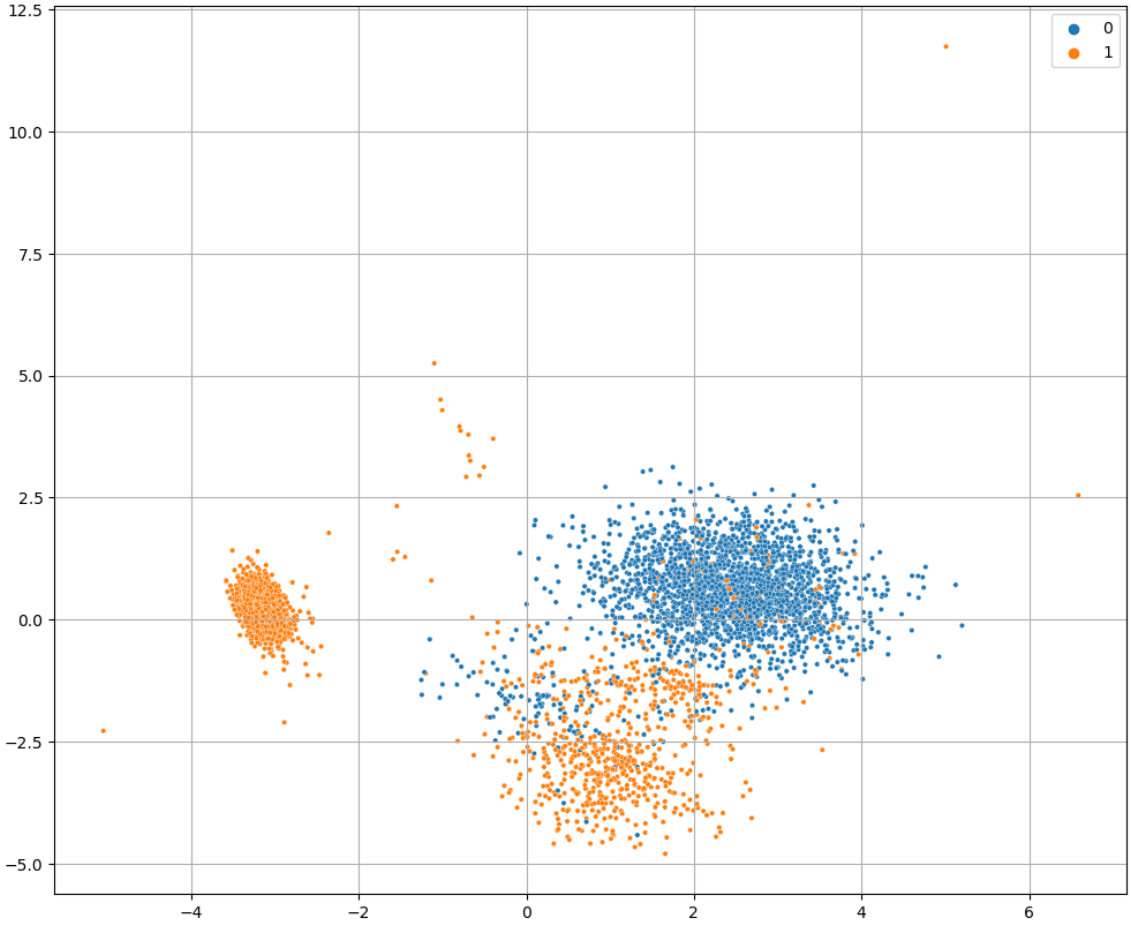


Figura 14. Visualización latente predicha con un F1 score de 94 % utilizando una PCA y coloreando los grupos de datos normales y datos anómalos.

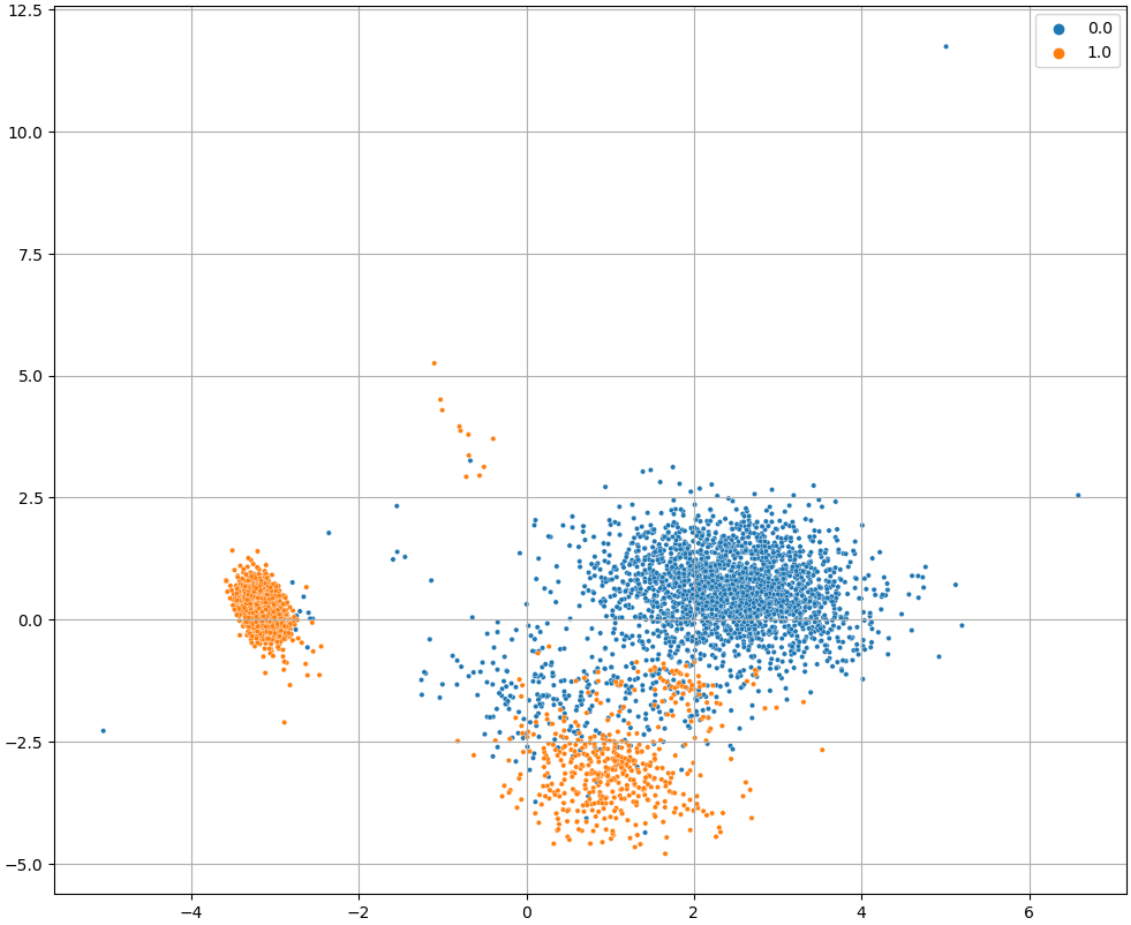
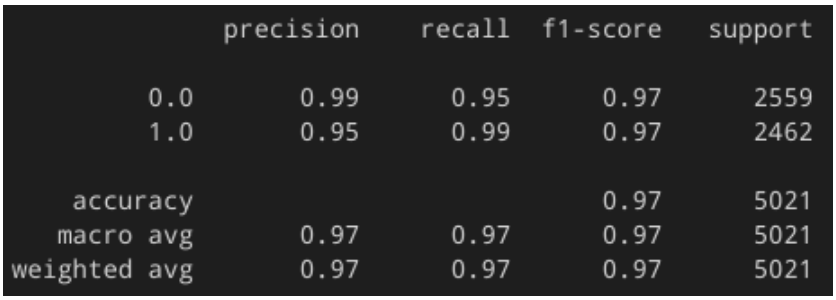


Figura 15. Visualización latente anotada para el ejemplo anterior, utilizando una PCA y coloreando los grupos de datos normales y datos anómalos.



	precision	recall	f1-score	support
0.0	0.99	0.95	0.97	2559
1.0	0.95	0.99	0.97	2462
accuracy			0.97	5021
macro avg	0.97	0.97	0.97	5021
weighted avg	0.97	0.97	0.97	5021

Figura 16. Métricas reportadas directamente desde la salida del código con el modelo entrenado con 500 epochs.

pérdida, en donde se tienen los siguientes reportes:

- Loss para 100 epoch fue de 1.416

- Loss para 200 epoch fue de 1.225

- Loss para 300 epoch fue de 1.092

- Loss para 500 epoch fue de 1.087

- Loss para 500 epoch fue de 1.085

Donde se evidencia que ya el modelo está lo suficiente ajustado ya que la función de pérdida deja de minimizarse mucho.

Con el modelo entrenado con 500 epochs se obtuvo el reporte:

	Precisión	Sensibilidad	F1-Score
Normal	0.99	0.95	0.97
Anomalía	0.95	0.99	0.97
Promedio	0.97	0.97	0.97

Tabla 2. Métricas reportadas para el modelo entrenado con 500 epochs y referenciado en la figura 16.

Para este entrenamiento también se hizo una visualización del espacio latente, presentada en la figura 18 y la figura 17.

Cabe resaltar que las representaciones en las imágenes corresponden al espacio latente y a los datos representados en aquel espacio latente, por lo cual, este espacio latente cambia cada vez que se reajuste el modelo, por ejemplo en el caso anterior que se aumentó el número de epochs del entrenamiento, lo cual varió el valor de la función de pérdida.

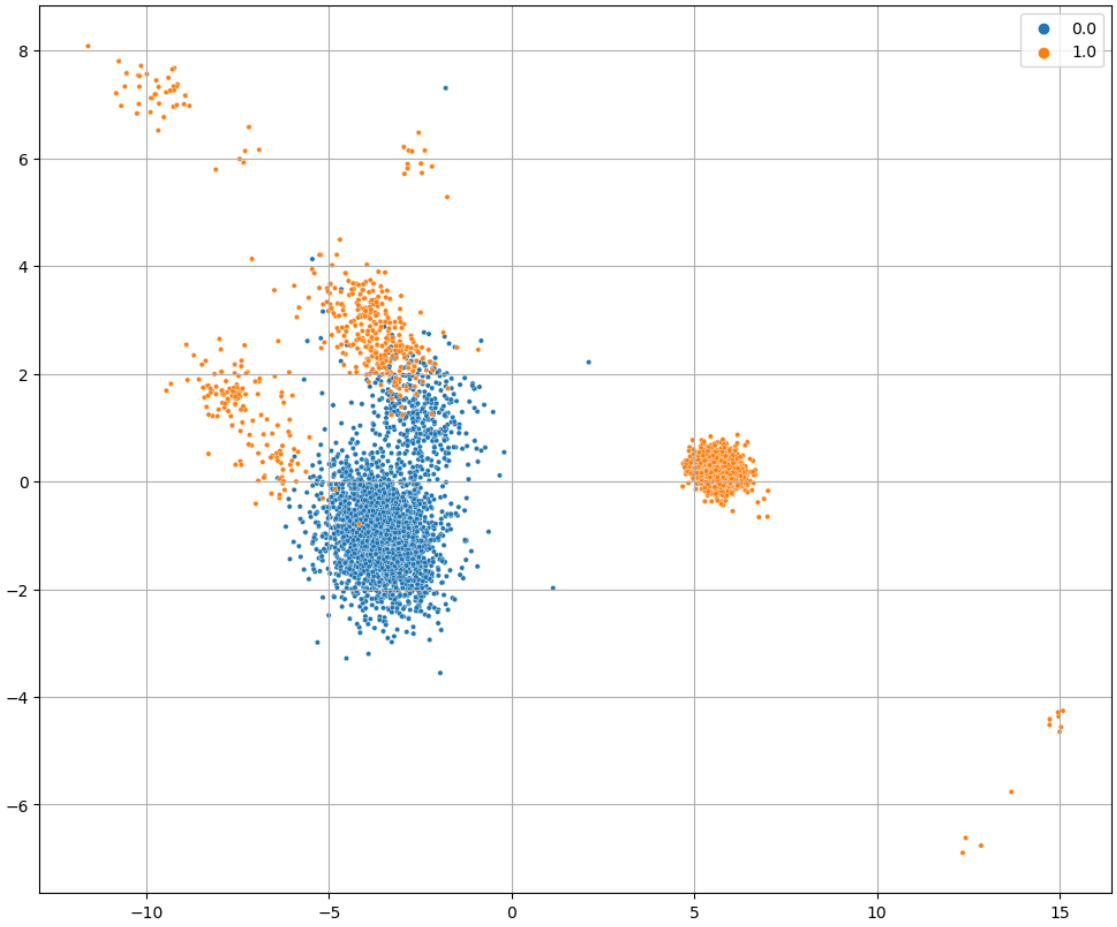


Figura 17. Visualización latente predicha con un F1-score de 97 %.

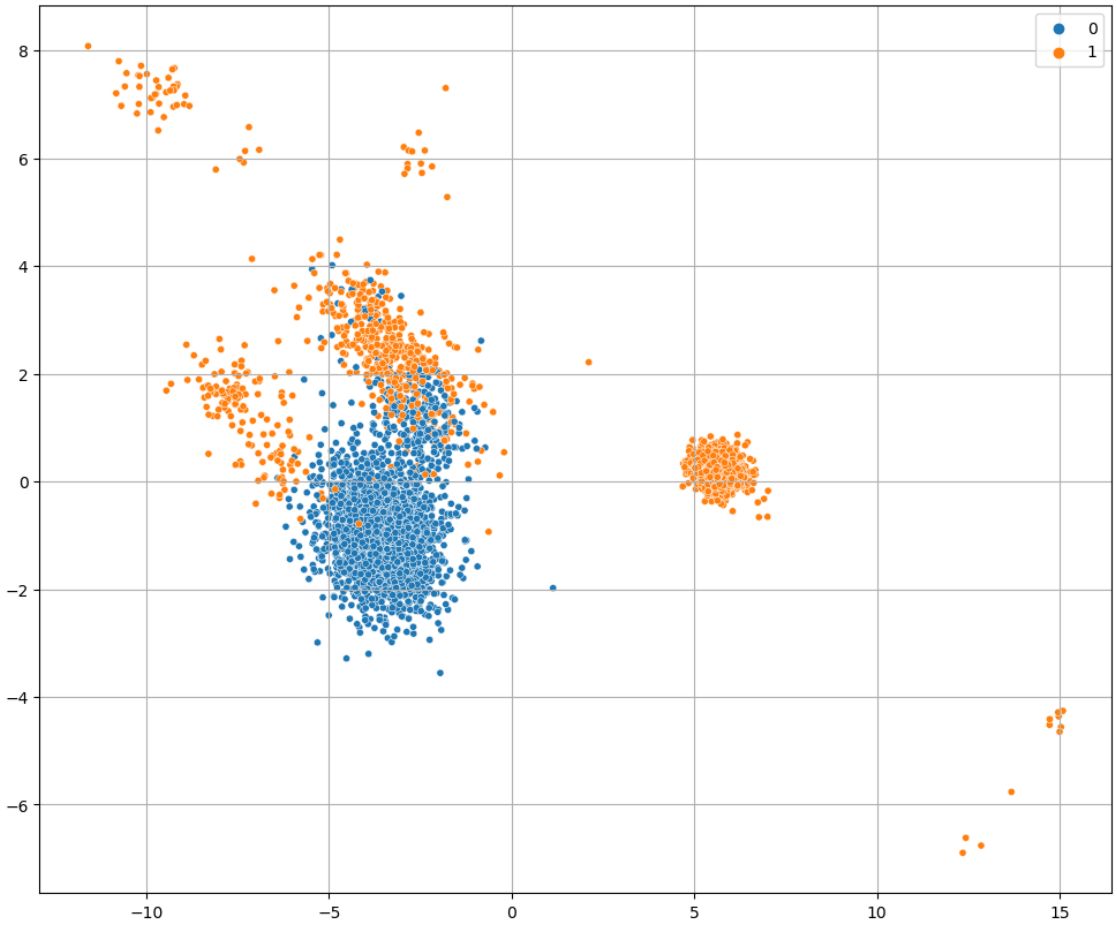


Figura 18. Visualización latente anotada para el anterior ejemplo.

El modelo alcanzó buenos niveles de precisión, sensibilidad y F1-score. La función de pérdida se estabilizó después de 500 epochs, indicando que el modelo presentó un buen ajuste a la hora de representar datos normales y por tanto en la reconstrucción de entradas anómalas se encontraba fuera de la distribución los datos normales y por tanto su clasificación como series que contenían potenciales fallos. Las visualizaciones del espacio latente apoyan estos hallazgos, mostrando una clara separación entre datos normales y anómalos, aunque con cierta superposición. Esto resalta la importancia de combinar métricas cuantitativas con representaciones gráficas para obtener una evaluación integral del rendimiento del modelo.

8. CONCLUSIONES Y RECOMENDACIONES

La administración de sistemas HPC enfrenta desafíos significativos debido a su creciente complejidad. Las técnicas tradicionales de monitoreo son a menudo insuficientes para manejar esta complejidad, lo que ha llevado a la exploración de modelos de aprendizaje profundo, como los autoencoders variacionales, que pueden detectar patrones anómalos en grandes volúmenes de datos operativos. Estos modelos proporcionan una detección de fallos más rápida y precisa, mejorando la resiliencia del sistema y reduciendo la necesidad de intervenciones manuales.
La implementación de un enfoque automatizado para la detección de anomalías no solo optimiza el tiempo de respuesta frente a fallos, sino que también permite a los administradores centrarse en tareas más estratégicas. Este estudio ha desarrollado y evaluado un modelo matemático basado en datos que utiliza aprendizaje profundo para la detección y manejo de fallos en sistemas HPC, demostrando un rendimiento satisfactorio y subrayando su potencial para integrarse en un marco de trabajo que soporte eficazmente la administración de estos sistemas.

8.1. Conclusiones

- En la revisión del estado del arte se observaron diferentes técnicas utilizadas para el monitoreo del estado del sistema, determinando que la implementación de un sistema de monitoreo robusto es necesario para que a través del uso de los logs se pueda llevar a cabo una detección temprana de condiciones anómalas y fallos potenciales, subrayando la importancia de una identificación proactiva de potenciales problemas para mantener la continuidad operativa del sistema.

- El uso de modelos de aprendizaje profundo está fuertemente relacionado en como se aborde el problema y factores como la extracción y representación de características. En este caso, la utilización de un autoencoder variacional fue particularmente útil para detectar patrones anómalos en las secuencias de eventos que potencialmente puediesen representar errores o

fallos que afecten la operación del sistema.

- Adoptar un enfoque automatizado para la detección de anomalías es una herramienta útil para la detección de errores ya permitiría reducir el número de intervenciones manuales realizadas por los administradores del sistema. Esto hace que la operación de la infraestructura sea más resiliente y tolerante a fallos, reduciendo el tiempo empleado en el relanzamiento de procesos. El modelo propuesto demuestra la eficacia del aprendizaje profundo en la incorporación de procesos automáticos de monitoreo y tolerancia a fallos.

- El modelo implementado mostró un rendimiento satisfactorio el cual promete que su incorporación en el análisis de fallas. La evaluación del modelo no solo confirmó su eficacia, sino que también destacó su potencial para integrarse en un marco de trabajo que soporte actividades de administración de un sistema HPC. Este enfoque brindaría una mayor eficiencia operativa y un mejor manejo de los fallos, haciendo más productiva la administración y mantenimiento de una infraestructura HPC.

8.2. Recomendaciones

- Considerando los resultados obtenidos en este estudio, donde se ha desarrollado un modelo predictivo efectivo para detectar patrones de anomalías en sistemas HPC, se recomienda integrar este modelo dentro de los componentes encargados de la recuperación del sistema en caso de fallo. Esta integración permitirá no solo detectar posibles fallas con anticipación, sino también tomar medidas proactivas, como la creación de checkpoints, que ayuden a garantizar la continuidad de las operaciones y minimizar el impacto de los fallos. Esta medida no solo mejoraría la fiabilidad del sistema, sino que también reduciría los tiempos de inactividad y minimizaría el impacto de posibles fallos en las operaciones críticas de las aplicaciones en ejecución.

- Como trabajo futuro se recomienda el despliegue e integración de este modelo predictivo a un software de monitoreo, para esto es necesario seleccionar la herramienta de monitoreo

más compatible, además del levantamiento de requerimientos, el diseño e implementación de interfaces de comunicación, también llevar a cabo pruebas en un entorno controlado para validar su integración. Además, será crucial optimizar continuamente el modelo basándose en el feedback obtenido durante su uso en el entorno real. Este esfuerzo mejorará significativamente la detección de fallos en tiempo real y establecerá una base robusta para futuras investigaciones en la tolerancia a fallos en sistemas HPC.

BIBLIOGRAFÍA

Baig, Mirza M., Mian.M. Awais y El-Sayed M. El-Alfy. «AdaBoost-based artificial neural network learning». En: *Neurocomputing* 248 (2017), págs. 120-126. DOI: 10.1016/J.NEUCOM.2017.02.077 (vid. pág. 18).

Bank, Dor, Noam Koenigstein y Raja Giryes. *Autoencoders*. 2021. arXiv: 2003.05991 [cs.LG] (vid. págs. 20, 21).

Benoit, Anne, Saurabh K. Raina e Yves Robert. «Efficient checkpoint/verification patterns». En: *International Journal of High Performance Computing Applications* 31.1 (2017), págs. 52-65. DOI: 10.1177/1094342015594531 (vid. pág. 31).

Benoit, Anne et al. «Multi-level checkpointing and silent error detection for linear workflows». En: *Journal of Computational Science* 28 (2018), págs. 398-415. DOI: 10.1016/j.jocs.2017.03.024 (vid. pág. 31).

Bosilca, George et al. «A failure detector for HPC platforms». En: *International Journal of High Performance Computing Applications* 32.1 (2018), págs. 139-158. DOI: 10.1177/1094342017711505 (vid. pág. 32).

Butler, Michael et al. *Proceedings of the Workshop on Methods, Models and Tools for Fault Tolerance (MeMToFT 2007)*. Vol. 5454. Information Society Tecnologies, jul. de 2007. DOI: 10.1007/978-3-642-00867-2 (vid. pág. 26).

Casas, Marc, Wilfried N. Gansterer y Elias Wimmer. «Resilient gossip-inspired all-reduce algorithms for high-performance computing: Potential, limitations, and open questions». En: *In-*

ternational Journal of High Performance Computing Applications 33.2 (2019), págs. 366-383. DOI: 10.1177/1094342018762531 (vid. pág. 31).

Chen, Zhuangbin y Michael R Lyu. *Experience Report : Deep Learning-based System Log Analysis for Anomaly Detection*. Inf. téc. 2022. DOI: https://doi.org/10.1145/1122445.1122456. arXiv: arXiv:2107.05908v2 (vid. pág. 34).

David L. Olson, Dursun Delen. *Advanced Data Mining Techniques*. 1st ed. Springer, 2008 (vid. pág. 37).

Egwutuoha, Ifeanyi P. et al. «A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems». En: *Journal of Supercomputing* 65.3 (2013), págs. 1302-1326. DOI: 10.1007/s11227-013-0884-0 (vid. pág. 25).

Farzad, Amir y T Aaron Gulliver. «Unsupervised log message anomaly detection». En: *ICT Express* 6.3 (2020), págs. 229-237. DOI: 10.1016/j.icte.2020.06.003 (vid. pág. 33).

Flach, Peter y Meelis Kull. «Precision-Recall-Gain Curves: PR Analysis Done Right». En: *Advances in Neural Information Processing Systems*. Ed. por C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015 (vid. págs. 28, 29).

Gainaru, Ana et al. «Failure prediction for HPC systems and applications: Current situation and open issues». En: *International Journal of High Performance Computing Applications* 27.3 (2013), págs. 273-282. DOI: 10.1177/1094342013488258 (vid. pág. 13).

Goodfellow, Ian, Yoshua Bengio y Aaron Courville. *Deep Learning*. 2016, pág. 775 (vid. págs. 16, 18).

Guo, Luanzheng, Dong Li e Ignacio Laguna. «PARIS: Predicting application resilience using machine learning». En: *Journal of Parallel and Distributed Computing* 152 (2021), págs. 111-124. DOI: 10.1016/j.jpdc.2021.02.015. arXiv: 1812.02944 (vid. pág. 32).

Hand, David J., Peter Christen y Nishadi Kirielle. «F*: an interpretable transformation of the F-measure». En: *Machine Learning* 110.3 (2021), págs. 451-456. DOI: 10.1007/s10994-021-05964-1 (vid. pág. 29).

He, Shilin et al. «Experience Report : System Log Analysis for Anomaly Detection». En: *2016 IEEE 27th International Symposium on Software Reliability Engineering*. 2016. DOI: 10.1109/ISSRE.2016.21 (vid. págs. 33, 34).

Heldens, Stijn et al. «The Landscape of Exascale Research: A Data-Driven Literature Analysis». En: *ACM Computing Surveys* 53.2 (2020). DOI: 10.1145/3372390 (vid. pág. 13).

Herzog, Jared. «Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman». eng. En: *Software engineering notes* 40.1 (2015), págs. 51-52 (vid. págs. 25, 27, 28).

High Performance Computing Chile, National Laboratory for. *Laboratorio Nacional de Computación de Alto Rendimiento (NLHPC)*. URL: https://www.nlhpc.cl/ (visitado 14-02-2024) (vid. pág. 13).

Ibtesham, Dewan, Kurt B. Ferreira y Dorian Arnold. «A checkpoint compression study for high-performance computing systems». En: *International Journal of High Performance Computing Applications* 29.4 (2015), págs. 387-402. DOI: 10.1177/1094342015570921 (vid. pág. 31).

Inc, SAS Institute. *SAS® Enterprise Miner™ 15.1: Reference Help*. URL: https://documentation.sas.com/?docsetId=emref&docsetTarget=n061bzurmej4j3n1jnj8bbjjm1a2.htm&docsetVersion=15.1&locale=en (visitado 17-09-2019) (vid. pág. 37).

Kwon, Donghwoon et al. «A survey of deep learning-based network anomaly detection». En: *Cluster Computing* 22.s1 (2019), págs. 949-961. DOI: 10.1007/s10586-017-1117-8 (vid. págs. 33, 34).

Landauer, Max et al. «Deep learning for anomaly detection in log data: A survey». En: *Machine Learning with Applications* 12 (2023), pág. 100470. DOI: https://doi.org/10.1016/j.mlwa.2023.100470 (vid. pág. 34).

Le, Van-hoang y Hongyu Zhang. «Log-based Anomaly Detection with Deep Learning : How Far Are We ?» En: *44th International Conference on Software Engineering (ICSE '22), May 21â•fi29, 2022, Pittsburgh, PA, USA*. Association for Computing Machinery, 2022. DOI: 10.1145/3510003.3510155. arXiv: arXiv:2202.04301v2 (vid. pág. 34).

Learn, Scikit. *Underfitting vs. Overfitting*. URL: https://scikit-learn.org/0.15/auto_examples/plot_underfitting_overfitting.html (visitado 14-02-2019) (vid. pág. 17).

Li, Jingbo et al. «GARLSched: Generative adversarial deep reinforcement learning task scheduling optimization for large-scale high performance computing systems». En: *Future Generation Computer Systems* 135 (2022), págs. 259-269. DOI: 10.1016/j.future.2022.04.032 (vid. pág. 32).

Li, Zhimin et al. «SpotSDC: Revealing the Silent Data Corruption Propagation in High-Performance Computing Systems». En: *IEEE Transactions on Visualization and Computer Graphics* 27.10 (2021), págs. 3938-3952. DOI: 10.1109/TVCG.2020.2994954 (vid. pág. 32).

Lima, André Luis da Cunha Dantas et al. «Smart predictive maintenance for high-performance computing systems: a literature review». En: *Journal of Supercomputing* 77.11 (2021), págs. 13494-13513 DOI: 10.1007/s11227-021-03811-7 (vid. pág. 32).

Linnainmaa, Seppo. «The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors». Tesis doct. Master's Thesis (in Finnish), Univ. Helsinki, 1970 (vid. pág. 19).

Losada, Nuria. «Application-level Fault Tolerance and Resilience in HPC Applications». Tesis doct. Universidade da Coruña, 2018, pág. 145 (vid. pág. 28).

Marbán, Óscar, Gonzalo Mariscal y Javier Segovia. *A Data Mining & Knowledge Discovery Process Model*. Ed. por Julio Ponce y Adem Karahoca. February. I-Tech, 2009, pág. 436. DOI: 10.5772/6438. arXiv: 1411.2705 (vid. pág. 37).

Matignon, Randall. *Neural Network Modeling Using Sas Enterprise Miner*. Authorhouse, 2005 (vid. pág. 37).

Nielsen, Michael A. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015 (vid. pág. 19).

Santander, Universidad Industrial de. *Supercomputación y Cálculo Científico*. URL: https://www.sc3.uis.edu.co/ (visitado 14-02-2024) (vid. pág. 13).

Schroeder, Bianca y Garth A. Gibson. «A large-scale study of failures in high-performance computing systems». En: *Proceedings of the International Conference on Dependable Systems and Networks* 2006 (2006), págs. 249-258. DOI: 10.1109/DSN.2006.5 (vid. pág. 13).

Shahzad, Faisal et al. «CRAFT: A library for easier application-level Checkpoint/Restart and Automatic Fault Tolerance». En: *IEEE Transactions on Parallel and Distributed Systems*

30.3 (2018), págs. 501-514. DOI: 10.1109/TPDS.2018.2866794. arXiv: 1708.02030 (vid. pág. 31).

Sinha, Rohit et al. «Anomaly Detection Using System Logs: A Deep Learning Approach». En: *International Journal of Information Security and Privacy* 16.1 (), págs. 1-15. DOI: 10.4018/IJISP.285584 (vid. pág. 34).

SLURM. *Simple Linux Utility for Resource Management*. URL: https://slurm.schedmd.com/overview.html (visitado 14-02-2024) (vid. pág. 39).

Wirth, Rüdiger. «CRISP-DM : Towards a Standard Process Model for Data Mining». En: *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining* 24959 (1995), págs. 29-39. DOI: 10.1.1.198.5133 (vid. pág. 36).

Yadav, Rakesh Bahadur. «A Survey on Log Anomaly Detection using Deep Learning». En: (2020), págs. 1215-1220 (vid. pág. 34).

Yang, Yuanhao y Hong Shen. «Deep Reinforcement Learning Enhanced Greedy Optimization for Online Scheduling of Batched Tasks in Cloud HPC Systems». En: *IEEE Transactions on Parallel and Distributed Systems* 33.11 (2022), págs. 3003-3014. DOI: 10.1109/TPDS.2021.3138459 (vid. pág. 33).