Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

Parametrización de redes de regulación genética mediante técnicas de aprendizaje máquina

Tesis presentada al

Posgrado en Física Aplicada

como requisito parcial para la obtención del grado de

Maestro en Ciencias

por

Lic. Rafael Sánchez Cedillo

Asesorado por

Dr. Jorge Velázquez Castro
Dr. Benito De Celis Alonso

Puebla Pue.
Diciembre de 2023

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

Parametrización de redes de regulación genética mediante técnicas de aprendizaje máquina

Tesis presentada al

Posgrado en Física Aplicada

como requisito parcial para la obtención del grado de

Maestro en Ciencias

por

Lic. Rafael Sánchez Cedillo

Asesorado por

Dr. Jorge Velázquez Castro
Dr. Benito De Celis Alonso

Puebla Pue.
Diciembre de 2023

Título: Parametrización de redes de regulación genética mediante técnicas de aprendizaje máquina

Estudiante: Lic. Rafael Sánchez Cedillo

COMITÉ

Dr. Javier Miguel Hernández López
Presidente

Dra. Beatriz Bonilla Capilla
Secretario

Dr. Jacobo Oliveros Oliveros
Vocal

Dr. Wuiyebaldo Fermín Guerrero Sánchez
Suplente

Dr. Jorge Velázquez Castro
Asesor

Dr. Benito De Celis Alonso
Segundo Asesor

Agradecimientos

En este significativo momento de mi vida académica, me siento profundamente afortunado y agradecido por el amor y apoyo que he recibido de tantas personas especiales. Primero y sobre todo, quiero expresar mi más sincero agradecimiento a mi familia: mi fuente de amor, fuerza y sabiduría.

A mi madre y a mi padre, por su apoyo incondicional y su guía invaluable; han sido mi faro en los momentos de incertidumbre y mi celebración en los momentos de éxito. A mi hermano, por sus consejos de vida y por caminar junto a mí en este camino, compartiendo experiencias y aprendizajes. Y a mi abuela, por su compañía y las historias que han enriquecido mi jornada con sabiduría y calidez.

Cada uno de ustedes ha jugado un papel vital en mi desarrollo, no solo como profesional sino como persona. Su amor y apoyo han sido las bases sobre las cuales he construido mis sueños y aspiraciones. Gracias por creer en mí, por estar a mi lado y por ser mi inagotable fuente de inspiración y motivación.

A mi querida perrita Nicky, quien ha sido más que una mascota para mí en este viaje académico. Su presencia constante y su inquebrantable lealtad fueron el consuelo en las largas noches de estudio y reflexión. Agradezco cada momento en que, con su mirada atenta y su compañía silenciosa, me recordó el valor de la perseverancia y la importancia de las pequeñas alegrías diarias. Este logro también lleva impresa tu huella, compañera incondicional.

Un reconocimiento especial es merecido para mis asesores, el Dr. Jorge Velázquez Castro y el Dr. Benito de Celis Alonso, por su guía experta, su apoyo constante y su comprensión en los momentos más cruciales de esta tesis. Su sabiduría y dedicación han sido fundamentales no solo en la conformación de este trabajo, sino también en mi crecimiento académico y personal.

Quiero agradecerles especialmente por su paciencia, sus valiosos consejos y su compañerismo a lo largo de este proceso. Su capacidad de entender y guiar, incluso en los momentos más desafiantes, ha sido un pilar en la conclusión exitosa de esta tesis. Estoy eternamente agradecido por su apoyo y por enseñarme el verdadero significado de la perseverancia y la excelencia académica.

Al cuerpo académico de Física Médica y al Posgrado en Física Aplicada, les agradezco profundamente por abrirme las puertas a un mundo de conocimiento y por brindarme la oportunidad de crecer y aprender en un ambiente tan estimulante y desafiante.

Elsa Biffano ha sido un pilar imortante en este proceso, brindándome apoyo, aliento y sabiduría cuando más lo necesitaba. Su presencia ha sido una luz en los momentos más desafiantes de esta travesía.

A mis amigos, compañeros de alma y aventuras, les debo un agradecimiento especial. Han sido

mi red de apoyo, mis confidentes, y la fuente de innumerables momentos de alegría y desahogo. Su amistad y apoyo incondicional han sido un tesoro invaluable, especialmente en los momentos de mayor presión y estrés.

A Francisco Valerio, un compañero de viaje en este camino académico, le agradezco su amistad, su compañerismo y su inquebrantable apoyo. Juntos hemos compartido desafíos y éxitos, y su presencia ha sido un factor clave en mi travesía.

Agradezco profundamente al Laboratorio Nacional de Supercómputo del Sureste de México de la Benemérita Universidad Autónoma de Puebla (BUAP) por proporcionar los recursos y apoyo necesarios para realizar los cálculos de este trabajo, dentro del marco del proyecto número 202204084C. El uso de sus instalaciones de supercómputo ha permitido el desarrollo y la validación de los modelos de redes neuronales implementados en esta investigación. Su contribución ha sido vital para avanzar en nuestro entendimiento de la red de regulación genética descrita por nuestro sistema de ecuaciones diferenciales. Esta colaboración subraya la importancia de las infraestructuras de supercómputo en la vanguardia de la investigación científica.

Por último, pero no menos importante, mi agradecimiento al Consejo Nacional de Ciencia y Tecnología (CONACyT) por su apoyo económico, que ha sido esencial para la realización de mis estudios y este proyecto. Su contribución ha sido fundamental en mi camino hacia la realización profesional y personal.

A todos ustedes, gracias desde lo más profundo de mi corazón.

Antes que el amor, el dinero, la fe, la fama y la justicia, dadme la verdad

Índice general

Índice de figuras

Índice de tablas

Resumen

In this study, the task of identifying parameters in genetic regulatory networks is addressed, focusing the study on the Toggle Switch model. This work has significant importance in the field of synthetic biology, as a precise understanding of these parameters is essential for manipulating and controlling complex biological systems. Such manipulation could lead to advances in gene therapies, bioproduct development, and understanding of fundamental biological mechanisms.

Identifying parameters in differential equation systems, which are at the heart of these models, presents unique challenges. These systems are inherently nonlinear and their behavior critically depends on multiple interconnected variables, making parameter identification complex and computationally intensive. In addition, the variability and noise inherent in biological experimental data add additional layers of difficulty.

Our method is to create an extensive synthetic database, reflecting a variety of potential behaviors of the biological system. This database includes both the evaluation of vector fields and the analysis of trajectories, providing a comprehensive perspective of the system's dynamics. We implement three types of neural networks - dense, convolutional, and a dense one with a custom cost function - to determine which offers the best performance in parameter identification.

Furthermore, to test the robustness of our approach, we introduce noise into our data, representing the common uncertainty and variability in real biological systems. This incorporation of noise allows us to assess how our neural networks adapt and perform under more realistic and challenging conditions.

The goal of this research is to develop a tool that not only improves our understanding of genetic regulatory systems but also enhances our ability to influence them in a precise and effective manner. This advancement represents a significant step towards engineering biological systems at a more refined and controlled level, opening new possibilities in biotechnology and medicine.

Keywords: Dynamic System, Genetic Regulatory Networks, Parameter Identification, Inverse Problems, Stochastic Processes, Neural Networks

En este estudio, se aborda la tarea de identificar parámetros en redes de regulación genética, centrando el estudio en el modelo de Toggle Switch. Este trabajo tiene una importancia significativa en el campo de la biología sintética, ya que una comprensión precisa de estos parámetros es esencial para manipular y controlar sistemas biológicos complejos. Dicha manipulación podría conducir a avances en terapias genéticas, desarrollo de bioproductos y comprensión de mecanismos biológicos fundamentales.

Identificar parámetros en sistemas de ecuaciones diferenciales, que son el corazón de estos modelos, presenta desafíos únicos. Estos sistemas son inherentemente no lineales y su comportamiento depende críticamente de múltiples variables interconectadas, lo que hace que la identificación de parámetros sea compleja y computacionalmente intensiva. Además, la variabilidad y el ruido inherentes a los datos experimentales biológicos agregan capas adicionales de dificultad.

Nuestro método es crear una base de datos sintética extensa, que refleja una variedad de comportamientos potenciales del sistema biológico. Esta base de datos incluye tanto la evaluación de campos vectoriales como el análisis de trayectorias, proporcionando una perspectiva integral de la dinámica del sistema. Implementamos tres tipos de redes neuronales - densa, convolucional y una densa con una función de costo personalizada - para determinar cuál ofrece el mejor rendimiento en la identificación de parámetros.

Además, para probar la robustez de nuestro enfoque, introducimos ruido en nuestros datos, representando la incertidumbre y variabilidad comunes en los sistemas biológicos reales. Esta incorporación de ruido nos permite evaluar cómo nuestras redes neuronales se adaptan y se desempeñan bajo condiciones más realistas y desafiantes.

El objetivo de esta investigación es desarrollar una herramienta que no solo mejore nuestra comprensión de los sistemas de regulación genética, sino que también potencie nuestra habilidad para influir en ellos de manera precisa y efectiva. Este avance representa un paso significativo hacia la ingeniería de sistemas biológicos a un nivel más refinado y controlado, abriendo nuevas posibilidades en la biotecnología y la medicina.

Palabras clave: *Sistema dinámico, Redes de Regulación Genética, Identificación de Parámetros,Problemas Invsersos, Procesos Estocásticos, Redes Neuronales*

Introducción

La biología sintética, un campo emergente que combina principios de la ingeniería, biología y ciencias computacionales, ha revolucionado nuestra comprensión y manipulación de sistemas biológicos. Este campo se enfoca en el diseño y construcción de componentes biológicos y sistemas que no existen en el mundo natural, lo que permite el desarrollo de nuevas aplicaciones en medicina, agricultura y biotecnología [1, 2]. La habilidad para diseñar organismos con funciones deseadas abre un vasto potencial para soluciones innovadoras a problemas globales como enfermedades, escasez de alimentos y desafíos ambientales [3].

Dentro de la biología sintética, el estudio de redes de regulación genética es fundamental. Estas redes, que describen cómo los genes interactúan y controlan la expresión genética, son esenciales para entender los procesos biológicos y cómo pueden ser alterados o controlados [4]. Las redes de regulación genética no sólo juegan un papel crucial en los procesos celulares normales, sino que también están implicadas en numerosas enfermedades, incluyendo cáncer y trastornos genéticos [5].

El modelado y análisis de estas redes permiten a los investigadores predecir cómo los cambios en ciertos genes afectarán a otros y a la célula en su conjunto. Esta capacidad es particularmente valiosa en la biología sintética, donde los científicos buscan crear sistemas biológicos con comportamientos predecibles y controlables [6]. Sin embargo, este es un área desafiante debido a la complejidad inherente de las redes biológicas y la necesidad de herramientas y técnicas sofisticadas para su estudio [7].

Uno de los mayores desafíos en la biología sintética y en el estudio de redes de regulación genética es la identificación precisa y eficiente de parámetros en dichos sistemas. Esta tarea es crítica ya que los parámetros en las redes de regulación genética dictan cómo los genes interactúan entre sí y con su entorno, influenciando directamente el comportamiento celular [8]. Una identificación precisa de estos parámetros es esencial para el modelado correcto de las redes genéticas, lo cual es fundamental para manipular efectivamente sistemas biológicos en aplicaciones como terapias genéticas y diseño de organismos sintéticos [9].

La complejidad de estos sistemas surge de su naturaleza intrínsecamente no lineal y de la interconexión de múltiples variables, lo que convierte a la identificación de parámetros en una tarea computacionalmente intensiva y matemáticamente compleja. Además, la variabilidad y el ruido en los datos experimentales biológicos añaden capas adicionales de dificultad, haciendo que los métodos tradicionales de análisis sean menos eficientes o incluso inadecuados [10, 11].

En este trabajo, abordamos este desafío utilizando técnicas avanzadas de aprendizaje automático, en particular, redes neuronales. Nuestro enfoque implica la creación de una base de datos sintética que refleje la variedad de comportamientos posibles de los sistemas biológicos y la aplicación de diferentes arquitecturas de redes neuronales para identificar los parámetros de manera más eficiente y precisa. Al implementar y comparar redes neuronales densas, convolucionales y redes densas con funciones de costo personalizadas, buscamos determinar cuál ofrece el mejor

rendimiento en la tarea de identificación de parámetros [12]. Este enfoque no solo mejora nuestra comprensión de los sistemas de regulación genética, sino que también potencia nuestra capacidad para influir en ellos de manera precisa y efectiva, representando un avance significativo en el campo de la biología sintética [13, 14].

La identificación de parámetros en redes de regulación genética ha sido un área de intensa investigación, con diversos enfoques desarrollados a lo largo de los años. Tradicionalmente, los métodos de análisis se han centrado en técnicas deterministas, como el ajuste de curvas y la minimización de errores cuadráticos, para estimar los parámetros de modelos basados en ecuaciones diferenciales [15]. Estos métodos, aunque eficaces en ciertas condiciones, a menudo enfrentan limitaciones cuando se trata de sistemas complejos y no lineales típicos de la biología [16].

Con el avance de la bioinformática, se han desarrollado métodos más sofisticados, como el análisis de redes bayesianas [17] y los algoritmos genéticos [18], que ofrecen una mayor flexibilidad y potencial para manejar la complejidad inherente a estos sistemas. Sin embargo, estos métodos pueden ser computacionalmente intensivos y a menudo requieren un gran volumen de datos para ser efectivos.

En el contexto de los datos estocásticos, los métodos tradicionales enfrentan desafíos aún mayores debido a la variabilidad y el ruido inherentes a los sistemas biológicos. Los enfoques estocásticos, como los métodos de Monte Carlo y las cadenas de Markov [19], se han utilizado para abordar estas limitaciones, permitiendo un mejor manejo de la incertidumbre y la variabilidad en los datos [20]. Sin embargo, estos métodos también pueden ser susceptibles a problemas de convergencia y eficiencia, especialmente en sistemas con un gran número de parámetros.

La integración de técnicas de aprendizaje automático, especialmente las redes neuronales, representa un enfoque innovador para superar estas limitaciones. A diferencia de los métodos tradicionales, las redes neuronales tienen la capacidad de aprender patrones complejos y no lineales directamente de los datos, lo que las hace particularmente adecuadas para modelar sistemas biológicos con datos estocásticos [12]. A pesar de su potencial, el desafío principal en la aplicación de redes neuronales radica en la necesidad de grandes cantidades de datos para el entrenamiento y la posibilidad de sobreajuste, especialmente en sistemas con una alta dimensionalidad de parámetros.

En el núcleo del estudio está el modelo de Toggle Switch, ejemplificando la interacción entre dos genes que se inhiben mutuamente. Este modelo es crucial para la biología sintética, ilustrando una regulación genética compleja. Las ecuaciones que definen este modelo son:

$$\frac{dX}{dt} = \frac{a_1}{1+Y^n} - d_1 X + b_1$$
$$\frac{dY}{dt} = \frac{a_2}{1+X^n} - d_2 Y + b_2$$

Aquí, X e Y representan las concentraciones de dos proteínas reguladoras, y los parámetros a_1, a_2, d_1, d_2, b_1, b_2, y n dictan las dinámicas del sistema.

La contribución de este estudio se encuentra en su enfoque interdisciplinario, combinando técnicas avanzadas de aprendizaje automático con problemas complejos de biología sintética. Esperamos que los resultados no solo ofrezcan insights sobre el modelo de Toggle Switch sino que también abran nuevas vías para la investigación en el modelado y análisis de sistemas biológicos complejos.

El capítulo inicial de la tesis proporciona un marco teórico y metodológico esencial para comprender el estudio. Se inicia con una introducción a la biología de sistemas, resaltando su enfoque integrador para analizar procesos biológicos complejos. Posteriormente, se profundiza en el estudio de redes de regulación genética, con un énfasis particular en sus modelos y la influencia crucial de las funciones de Hill en su modelado.

Un enfoque especial se da al modelo de Toggle Switch genético, descrito anteriormente y que es el modelo base de este trabajo. Esta sección subraya la complejidad y la interconexión inherentes en los sistemas biológicos. Además, se discute la importancia de los métodos numéricos en la modelización de dinámicas biológicas, destacando el uso del método de Runge-Kutta de quinto orden (RK5) y el método de Euler-Mayorana para resolver ecuaciones diferenciales estocásticas.

La sección final del capítulo se dedica a las redes neuronales, explicando sus componentes fundamentales y su aplicación en la biología de sistemas. Esta parte no solo contextualiza el uso de técnicas avanzadas de aprendizaje automático en nuestra investigación, sino que también establece la base para entender cómo estas herramientas pueden ser implementadas para abordar problemas complejos en biología sintética.

En esta etapa, introducimos Python y TensorFlow, destacando su relevancia en el estudio. Mencionamos específicamente cómo Python, por su versatilidad y amplia adopción en la comunidad científica, se convierte en un lenguaje clave para la implementación de nuestras soluciones de aprendizaje automático. TensorFlow, por su parte, es resaltado como un framework esencial que facilita la construcción y entrenamiento de modelos de redes neuronales, siendo una herramienta vital para nuestra investigación.

Además, se hace una mención breve pero esencial al software complementario que se utiliza en la investigación. Sin adentrarnos en detalles técnicos, subrayamos que la selección de estas herramientas de software es estratégica, con el objetivo de maximizar la eficiencia y efectividad en el análisis y modelado de sistemas biológicos complejos. Con esta discusión, el capítulo cierra estableciendo una base sólida y coherente para los desarrollos y análisis que se explorarán en los siguientes segmentos de la tesis.

El segundo capítulo de la tesis se enfoca en detallar la metodología empleada en este estudio. Se inicia con la elaboración de bases de datos tanto deterministas como estocásticas, incorporando soluciones numéricas y campos vectoriales vinculados al sistema de ecuaciones del modelo Toggle Switch. Se explican los modelos de redes neuronales utilizados, abarcando desde su diseño inicial hasta la implementación y optimización de hiperparámetros con herramientas como Optuna.

Este capítulo también abarca aspectos clave como la selección de métricas apropiadas, la elección de optimizadores, y la implementación de una tasa de aprendizaje adaptativa. Se describen métodos para la visualización de métricas y la evaluación del modelo con datos de prueba, enfocándose en la capacidad de las redes para predecir y analizar datos bajo diferentes escenarios. Se describe también el análisis estadístico final, tanto para los datos deterministas como estocásticos, que nos ayudará a evaluar nuestros modelos en la sección de resultados.

Se describe también el análisis de un modelo único de red neuronal, diseñado para procesar entradas de cualquier tamaño sin la necesidad de especificar sus dimensiones exactas. Se concluye esta sección evaluando el impacto de la reducción del tamaño de las bases de datos en la precisión de los coeficientes estimados. Especialmente relevante es la investigación sobre el tamaño mínimo de experimento necesario para garantizar la fiabilidad de nuestra metodología. Experimentamos con bases de datos más pequeñas, tanto para campos vectoriales como para trayectorias y tanto con ruido como sin ruido. Este análisis es crucial para comprender el impacto de la escasez de datos

en la precisión de los modelos y la adaptabilidad de nuestra metodología a diferentes contextos experimentales. En resumen, este capítulo proporciona una guía detallada sobre la metodología empleada, destacando su innovación y relevancia en el campo de la biología de sistemas y la regulación genética.

El tercer capítulo de la tesis se dedica a presentar y analizar los resultados obtenidos de la aplicación de los modelos neuronales. Aquí se comparan las métricas de los distintos modelos en contextos tanto deterministas como estocásticos, proporcionando una visión clara de su rendimiento en variadas condiciones. Se pone un énfasis particular en el análisis estadístico detallado, cuyo objetivo es evaluar la confiabilidad y precisión de las predicciones generadas por los modelos.

Este capítulo busca evidenciar la efectividad de los enfoques desarrollados en el estudio y subraya la robustez de los modelos neuronales en la tarea crítica de identificación de parámetros en los sistemas de regulación genética.

Finalmente, la sección de conclusiones resume los hallazgos clave, destacando las contribuciones del estudio al campo de la biología de sistemas y proponiendo direcciones futuras para la investigación. La tesis también incluye apéndices que proporcionan detalles adicionales sobre aspectos específicos del modelado y análisis realizados.

En conjunto, esta tesis ofrece una visión integral y detallada de la modelación y análisis de redes de regulación genética utilizando técnicas avanzadas de aprendizaje automático, con especial énfasis en el modelo Toggle Switch. A través de esta investigación, se busca no solo avanzar en el conocimiento teórico sino también proporcionar herramientas prácticas y métodos que puedan ser aplicados en el campo emergente de la biología sintética.

Objetivos

Objetivo General

Desarrollar una metodología utilizando redes neuronales para la identificación precisa de parámetros en modelos biológicos.

Objetivos Específicos

1. Implementar redes neuronales para determinar con exactitud los parámetros en el modelo Toggle Switch.
2. Evaluar la robustez de la metodología frente a datos tanto estocásticos como deterministas y en distintas resoluciones experimentales.
3. Optimizar las redes neuronales para mejorar la eficiencia y precisión en el análisis de datos genéticos.
4. Contribuir al campo de la biología sintética y terapia genética a través de la aplicación de los hallazgos obtenidos en la investigación.

Capítulo 1

Marco Teórico

1.1. Biología de Sistemas

1.1.1. Introducción

La biología de sistemas es un enfoque interdisciplinario que busca comprender y modelar los sistemas biológicos en su totalidad, desde las células individuales hasta los organismos completos. Se basa en la idea de que los organismos vivos son sistemas complejos y dinámicos, cuyas propiedades emergentes surgen de la interacción entre sus componentes [21]. Se busca entender cómo funcionan los sistemas biológicos a través del análisis de las interacciones entre genes, proteínas, metabolitos y otros componentes celulares. Se utilizan herramientas matemáticas, computacionales y experimentales para recopilar datos y desarrollar modelos que describan y expliquen los comportamientos observados [22].

Uno de los objetivos principales de la biología de sistemas es comprender cómo los componentes individuales de un sistema biológico se integran y coordinan para llevar a cabo funciones complejas. Se busca identificar las redes de interacción y retroalimentación que regulan los procesos biológicos y cómo estas redes se adaptan y responden a cambios en el entorno. Se basa en la idea de que los sistemas biológicos son dinámicos y pueden mostrar comportamientos emergentes que no pueden explicarse solo por el estudio de los componentes individuales. Se enfoca en el análisis de las propiedades y patrones que surgen de la interacción entre los componentes, así como en la comprensión de los principios generales que subyacen a estos sistemas.

Algunos de los enfoques y herramientas utilizados en la biología de sistemas incluyen la modelación matemática, el análisis de redes, la genómica, la proteómica, la metabolómica y la bioinformática. Estos enfoques permiten construir modelos y realizar simulaciones para predecir el comportamiento de los sistemas biológicos y generar hipótesis que pueden ser probadas experimentalmente [23]. Tiene aplicaciones en diversas áreas, como la medicina, la agricultura, la biotecnología y la biología sintética. Permite comprender mejor las enfermedades y desarrollar enfoques terapéuticos más precisos, diseñar sistemas biológicos personalizados y mejorar la producción de alimentos y productos químicos.

1.1.2. Redes de Regulación Genética

La célula, un sistema integrado y complejo que consta de miles de tipos diferentes de proteínas, funciona como una micro fábrica. Cada proteína, una molécula de tamaño nanométrico, realiza funciones específicas con una precisión asombrosa. Las células se adaptan a diversas situaciones,

produciendo diferentes proteínas según sea necesario. Por ejemplo, cuando detectan azúcar, las células comienzan a producir proteínas que transportan el azúcar al interior de la célula para su utilización. Similarmente, en respuesta a daños celulares, generan proteínas reparadoras. Esto demuestra la capacidad de las células para monitorear su entorno y regular la producción de proteínas de manera precisa. Las redes de transcripción juegan un papel principal en este proceso de regulación y procesamiento de la información.

Para representar y responder a estos variados estados ambientales, las células utilizan proteínas especiales llamadas factores de transcripción. Estos factores pueden cambiar rápidamente entre estados moleculares activos e inactivos con una frecuencia modulada por señales ambientales específicas.

Cada factor de transcripción activo se une al ADN para regular la tasa de transcripción de genes específicos (véase Figura 1.1). Estos genes se transcriben en ARN mensajero y luego se traducen en proteínas, que a su vez interactúan y modifican su entorno. La actividad de los factores de transcripción en una célula puede considerarse como una representación interna del entorno celular. Un ejemplo destacado es Escherichia coli (E. coli), una bacteria comúnmente utilizada en la investigación biológica y biotecnológica. E. coli mantiene una representación interna de su entorno con aproximadamente 300 factores de transcripción, que regulan la producción de alrededor de 4500 proteínas diferentes. E. coli, conocida por su rapidez de crecimiento y facilidad de manipulación genética, es un organismo modelo crucial en el estudio de la biología molecular y celular [24].

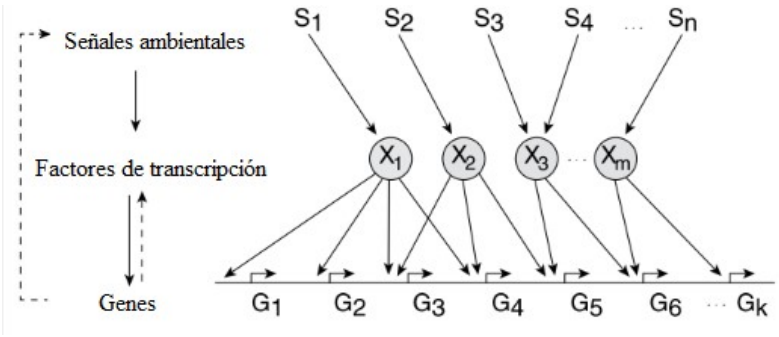


Figura 1.1: Representación de cómo los factores ambientales afectan a los factores de transcripción.

La representación interna de una célula a través de los factores de transcripción proporciona una descripción compacta de una gran cantidad de estímulos ambientales. Diferentes situaciones se resumen en la actividad de un factor de transcripción específico, que puede indicar condiciones como necesidad de nutrientes o daño en el ADN. De esta manera, los factores de transcripción regulan los genes objetivo, desencadenando respuestas proteicas adecuadas para cada situación específica.

Las interacciones entre los factores de transcripción y los genes se explican a través de las **redes de regulación genética**. En el contexto de estas redes, un gen se define como una porción de ADN cuya secuencia codifica la información necesaria para la síntesis de una proteína específica. Este proceso de fabricación de proteínas ocurre en dos fases: **transcripción** y **traducción**.

Durante la etapa de **transcripción**, la secuencia del gen es copiada en una molécula de ARN mensajero (ARNm) por una enzima llamada ARN polimerasa (ARNp). Este proceso se divide en diferentes pasos:

1. **Iniciación**: La ARN polimerasa se une a una secuencia específica de ADN conocida como promotor, marcando el inicio del gen.

2. **Elongación**: La ARN polimerasa desenrolla la doble hélice del ADN y comienza a sintetizar una hebra de ARNm usando uno de los hilos de ADN como plantilla. Este ARNm es una versión complementaria de la hebra de ADN, pero con uracilo (U) reemplazando a la timina (T).

3. **Terminación**: Una vez que la ARN polimerasa llega a una secuencia de terminación en el ADN, la producción de ARNm se detiene y la molécula de ARNm se libera.

Posteriormente, en la etapa de **traducción**, el ARNm es usado como plantilla para ensamblar una proteína específica. De igual manera este proceso se divide en pasos:

1. **Iniciación**: La molécula de ARNm se une a un ribosoma. Un codón de inicio en el ARNm (generalmente AUG, que codifica para el aminoácido metionina) señala el inicio de la traducción.

2. **Transcripción**: Cada codón del ARNm especifica un aminoácido particular. Un conjunto de moléculas de ARN de transferencia (ARNt) transporta los aminoácidos correctos al ribosoma, donde se unen en el orden especificado por el ARNm para formar una cadena de aminoácidos.

3. **Traslación**: Cuando el ribosoma llega a un codón de parada en el ARNm, la cadena de aminoácidos (ahora una proteína) se libera, y la traducción termina.

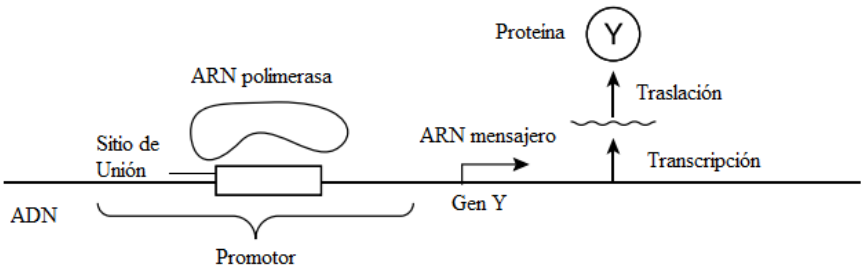


Figura 1.2: Proceso de creación de proteínas.

La tasa de producción de ARNm, y por ende la tasa de producción de genes por unidad de tiempo, es regulada por una sección de ADN precedente al gen conocida como **promotor**. Esta secuencia de ADN determina la afinidad química con la que la ARNp se une al ADN, estableciendo así la velocidad de transcripción del gen.

Si bien la ARNp actúa sobre todos los genes, las variaciones en la expresión de genes específicos son atribuibles a los factores de transcripción. Cada factor de transcripción ajusta la velocidad de transcripción de un conjunto específico de genes objetivo. Estos factores influyen en la tasa de transcripción al adherirse a sitios particulares en los promotores de los genes que regulan (Figura 1.3). Al unirse, alteran la probabilidad por unidad de tiempo de que la ARNp se acople al promotor y genere una molécula de ARNm. De este modo, los factores de transcripción impactan la rapidez con la que la ARNp inicia la transcripción del gen.

Los factores de transcripción pueden ejercer dos roles distintos: como **activadores**, incrementando la velocidad de transcripción de un gen (Figura 1.3 izquierda), o como **represores** que la

disminuyen (Figura 1.3 derecha).

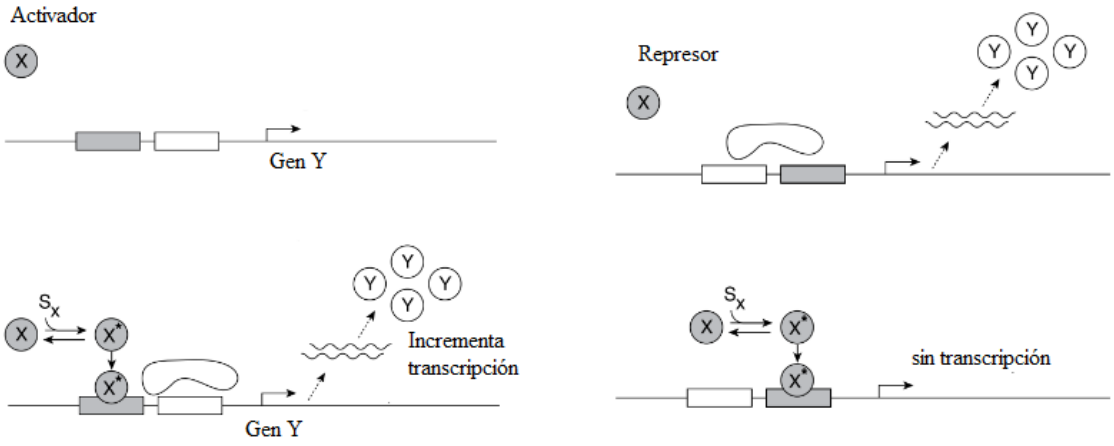


Figura 1.3: Factores de transcripción que actúan como activadores (izquierda) o represores (derecha).

Las proteínas que componen los factores de transcripción están codificadas por genes, los cuales, a su vez, están regulados por otros factores de transcripción. Esta interdependencia puede extenderse a otros factores de transcripción en un proceso de regresión sucesiva. Este complejo entramado de interacciones conforma una **red de regulación genética**, que engloba todas las interacciones reguladoras de la transcripción en una célula, cuyas entradas son señales cargadas de información procedente del entorno.

Cada señal puede ser una pequeña molécula o una modificación de proteína que afecta directamente la actividad de uno de los factores de transcripción. Frecuentemente, los estímulos externos desencadenan vías de transducción de señales bioquímicas que concluyen en una modificación química de factores de transcripción específicos. En otros sistemas, la señal puede ser tan sencilla como una molécula de azúcar o una hormona que ingresa a las células y se une directamente al factor de transcripción. Usualmente, las señales inducen un cambio físico en la forma de la proteína del factor de transcripción, lo que le permite adoptar un estado molecular activo. Por consiguiente, la señal S_x puede provocar que X transite rápidamente a su estado activo X^*, se adhiera al promotor del gen Y y modifique la tasa de transcripción, lo que resulta en una producción incrementada o disminuida de la proteína Y.

Las redes de regulación genética han sido objeto de intensa investigación en los últimos años. Estudios recientes han destacado la importancia de estas redes en la comprensión de la biología fundamental y su relevancia en las enfermedades humanas [25]. Las perturbaciones en estas redes han sido relacionadas con enfermedades genéticas y cáncer, donde cambios en la expresión génica pueden alterar la función celular y contribuir al desarrollo de estas patologías [26].

La medicina de precisión se ha beneficiado enormemente del estudio de las redes de regulación genética. La comprensión de cómo los genes interactúan y se regulan entre sí dentro de estas redes ha permitido identificar dianas terapéuticas específicas y mejorar la eficacia del tratamiento [27]. Este enfoque ha llevado a avances significativos en el diseño de terapias personalizadas y enfoques más precisos para el manejo de enfermedades complejas.

Además de su papel en enfermedades genéticas y cáncer, las redes de regulación genética también han revelado su implicación en enfermedades neurodegenerativas. Estudios han demostrado que los cambios en la expresión de ciertos genes dentro de estas redes pueden afectar la salud neuronal y contribuir al desarrollo de enfermedades como la enfermedad de Alzheimer y la enfermedad de Parkinson [28].

La respuesta inmunitaria también se ve profundamente influenciada por las redes de regulación genética. Estas redes determinan cómo las células del sistema inmunitario coordinan sus actividades y responden a estímulos específicos. Su comprensión es crucial en el contexto de enfermedades infecciosas, donde el sistema inmunitario desempeña un papel clave en la defensa del organismo [29].

Las redes de regulación genética representan una herramienta poderosa para comprender la complejidad de los sistemas biológicos y su relevancia en diversas enfermedades. Su estudio ha permitido avances significativos en la medicina de precisión, la comprensión de enfermedades genéticas y cáncer, enfermedades neurodegenerativas y la respuesta inmunitaria.

1.1.3. Modelación de Redes de Regulación Genética

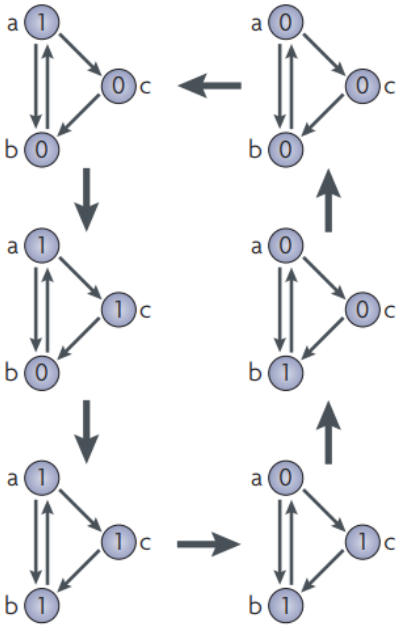
La modelación de redes de regulación genética es un campo fascinante y en constante desarrollo que busca comprender y predecir cómo los genes y los factores de transcripción interactúan para regular la expresión génica. Estas redes son complejas y dinámicas, y su estudio requiere herramientas matemáticas y computacionales para capturar la interacción entre múltiples componentes.

Uno de los métodos más prevalentes en el estudio de las redes de regulación genética es el uso de modelos basados en redes Booleanas, tal como se ilustra en la Figura 1.4. Estos modelos representan los genes como variables booleanas, que pueden adoptar dos estados: activado o desactivado. A través de operaciones lógicas, como AND, OR y NOT, se describen las interacciones y las influencias mutuas entre estos genes. Esta metodología fue inicialmente propuesta por Stuart Kauffman en 1969 [30] y ha demostrado ser una herramienta poderosa para el análisis de sistemas biológicos complejos.

En los modelos de redes Booleanas, la regulación genética se simplifica a interacciones binarias, lo que facilita la comprensión de los mecanismos subyacentes en las redes genéticas. A pesar de su simplicidad, estos modelos han sido extremadamente útiles para explorar los estados estables de las redes genéticas y entender cómo estas estructuras pueden influir en la dinámica celular. Por ejemplo, el análisis de los patrones de activación y desactivación de genes puede revelar cómo las células responden a diferentes estímulos o cómo se desarrollan ciertos estados patológicos.

Además, los modelos de redes Booleanas permiten a los investigadores simular y predecir el comportamiento de redes genéticas bajo diversas condiciones experimentales. Esto es especialmente relevante en el campo de la biología sintética y en el diseño de terapias génicas, donde comprender cómo se pueden manipular las redes genéticas para obtener resultados deseados es crucial.

Sin embargo, es importante destacar que, aunque los modelos Booleanos ofrecen una aproximación valiosa, también presentan limitaciones. La principal es su naturaleza binaria, que no permite capturar la gama completa de expresiones génicas y las complejidades de la regulación post-transcripcional. Por lo tanto, estos modelos se utilizan a menudo como un punto de partida para estudios más detallados que requieren enfoques más sofisticados, como los modelos basados en ecuaciones diferenciales o en redes neuronales.



Funciones de regulación

F_b (a,c)

a	c	b
0	0	0
0	1	1
1	0	0
1	1	1

F_a (b)

b	a
0	1
1	0

F_c (a)

a	c
0	0
1	1

Figura 1.4: Redes booleanas. Cada una de las entidades a, b y c de la red puede estar en el estado 0 o 1. Las transiciones de estado obedecen a las funciones de regulación que se muestran a la derecha, que describen las reglas del modelo. Por ejemplo, si a está en el estado 1 y c está en el estado 0, en el siguiente paso el estado de b será 0. Las flechas finas indican los reguladores de cada nodo. Los pasos de tiempo están representados por flechas gruesas. El estado global del modelo es la combinación de los tres estados de la entidad. El sistema recorre los seis estados globales. Una secuencia de estados globales consecutivos se llama trayectoria.

Además del análisis de redes Booleanas, otro enfoque ampliamente adoptado en la modelación de redes de regulación genética es el uso de ecuaciones diferenciales. Estos modelos matemáticos avanzados describen detalladamente cómo varían las concentraciones de ARN y proteínas con el tiempo. Consideran factores críticos como las tasas de síntesis y degradación de estas moléculas, así como las complejas interacciones entre genes y factores de transcripción [22]. Esta metodología es particularmente valiosa para simular y predecir la dinámica temporal de las redes genéticas, ofreciendo una visión más profunda y cuantitativa de los procesos biológicos.

Los modelos basados en ecuaciones diferenciales son especialmente útiles para entender cómo los cambios en ciertos parámetros pueden afectar el comportamiento global de la red. Por ejemplo, pueden ayudar a predecir cómo la alteración en la tasa de producción de una proteína específica podría influir en la estabilidad de todo el sistema.

Además de los modelos booleanos y de ecuaciones diferenciales, los enfoques estocásticos también juegan un papel fundamental en la modelación de redes de regulación genética. Estos modelos incorporan la variabilidad y la incertidumbre inherentes a los sistemas biológicos, capturando las fluctuaciones aleatorias en las interacciones moleculares. Esto es crucial para estudiar la heterogeneidad en las respuestas celulares y para entender cómo la variabilidad genética y ambiental puede influir en la función de las redes genéticas [31].

Es esencial reconocer que, aunque estos modelos proporcionan una comprensión profunda de los sistemas biológicos, son simplificaciones de la realidad. Por lo tanto, es crucial validarlos con

datos experimentales. Estas herramientas de modelación son invaluables para descifrar la complejidad y los patrones emergentes en la regulación génica. Ofrecen una base sólida para predecir comportamientos celulares y diseñar intervenciones efectivas en sistemas biológicos complejos.

1.1.4. Funciones de Hill

La función de Hill fue presentada por primera vez en 1910 en el trabajo seminal de A.V. Hill titulado 'The Possible Effects of the Aggregation of the Molecules of Haemoglobin on its Dissociation Curves' [32]. En este estudio, Hill describió cómo la unión y disociación de la hemoglobina se podían modelar mediante una función matemática.

Este enfoque innovador de Hill pronto encontró aplicaciones más allá de su contexto original. Investigadores en diversos campos de la biología y la medicina adoptaron la función de Hill para modelar la interacción entre ligandos y sus receptores. Con el avance de las técnicas experimentales y la acumulación de datos, se hizo evidente la utilidad de la función de Hill para describir fenómenos biológicos complejos.

En el ámbito de la regulación génica, las funciones de Hill adquirieron un papel crucial. Un ejemplo destacado es el modelo del operón lac propuesto por Jacques Monod y François Jacob en 1965 [33]. En este modelo, utilizaron funciones de Hill para explicar cómo el represor se une al promotor y regula la expresión génica en bacterias. Este trabajo no solo profundizó nuestra comprensión de la regulación génica, sino que también estableció un precedente para la aplicación de las funciones de Hill en este campo.

Con el tiempo, las funciones de Hill se han convertido en una herramienta esencial en disciplinas como la biología de sistemas y la biología sintética. Su capacidad para modelar respuestas no lineales y fenómenos de conmutación en sistemas biológicos ha sido invaluable. La simplicidad y versatilidad de estas funciones permiten a los científicos describir y predecir cómo los genes responden a diversos estímulos, facilitando así una mejor comprensión y manipulación de los sistemas biológicos.

Como ya hemos discutido anteriormente, los factores de transcripción pueden funcionar como activadores o represores en la regulación génica. Convencionalmente, un activador del gen X al gen Y se representa como $X \to Y$, mientras que un represor se ilustra con el símbolo $A \dashv B$.

Estas flechas no solo indican la dirección de la influencia, sino que también llevan números que corresponden a la magnitud de la interacción.La intensidad con la que un factor de transcripción afecta a un gen objetivo se cuantifica a través de una función de entrada específica.Para ilustrar esto, consideremos el caso de la tasa de producción de la proteína Y, la cual está bajo el control del factor de transcripción X. Cuando X regula a Y, el número de moléculas de proteína Y producidas por unidad de tiempo es una función de la concentración de X en su forma activa, X^*:

$$Y = f(X^*) \tag{1.1}$$

Habitualmente, la función de entrada $f(X^*)$ es una función monótona; es creciente cuando X es un activador y decreciente cuando X es un represor. Una forma común que describe de manera realista muchas funciones de entrada de genes es la **función de Hill**. La función de entrada de Hill para un activador es una curva que se eleva desde cero y se aproxima a un nivel máximo de saturación (Figura 1.5). Matemáticamente, se expresa como:

$$f(X^*) = \beta \frac{X^{*n}}{K^n + X^{*n}} \tag{1.2}$$

Donde:

- **K** es el coeficiente de activación y tiene unidades de concentración. Define la concentración de X activo necesaria para activar significativamente la expresión. Este valor es determinado por la afinidad química entre X y el sitio de unión en el promotor.

- β representa la máxima actividad promotora.

- **n** es el coeficiente de Hill, que indica la inclinación de la curva de respuesta. Un valor de $n > 1$ indica que la unión de cada unidad adicional de X^* aumenta significativamente la probabilidad de que las unidades subsiguientes se unan. Por otro lado, un valor de $n < 1$ sugiere que la unión de cada unidad adicional de X^* tiene un efecto menor en la probabilidad de unión de las siguientes, lo que conduce a una curva de respuesta más gradual. Un coeficiente de Hill de $n = 1$ indica que cada unidad de X^* tiene un efecto independiente, sin cooperación entre las unidades, lo que da como resultado una curva de respuesta de tipo hiperbólica estándar.

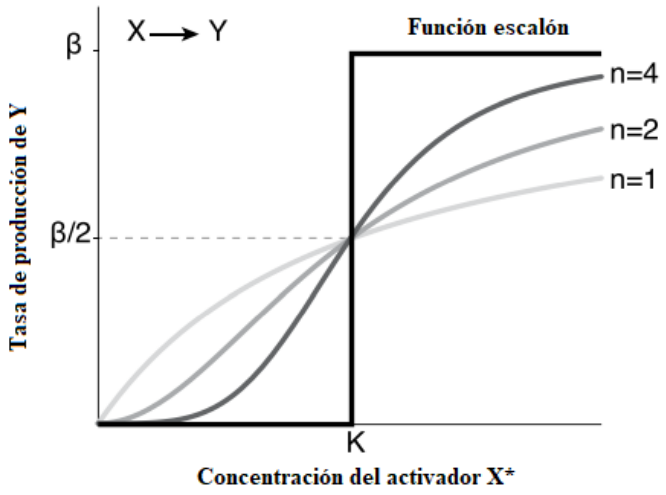


Figura 1.5: Función de Hill para un activador con distintos valores de n.

Para un represor $X \dashv Y$ la función de entrada de Hill describe una curva decreciente (Figura 1.6). . La ecuación que modela esta relación es similar a la de un activador, pero refleja el efecto inhibitorio del represor:

$$f(X^*) = \beta \frac{K^{*n}}{K^n + X^{*n}} \tag{1.3}$$

Las funciones de de Hill, tanto para activadores como para represores, oscilan entre una tasa de transcripción de cero y una tasa de transcripción máxima β. Sin embargo, muchos genes presentan un nivel de expresión mínimo distinto de cero, denominado nivel de expresión basal del gen. Dicho nivel basal se puede representar añadiendo a la función de entrada un término β_0.

La simplicidad de una función de Hill individual se transforma en complejidad al combinar múltiples funciones de Hill, permitiendo modelar interacciones génicas intrincadas. Por ejemplo, un gen regulado por varios factores de transcripción puede ser representado mediante la integración de distintas funciones de Hill, cada una correspondiente a un factor de transcripción diferente [22].

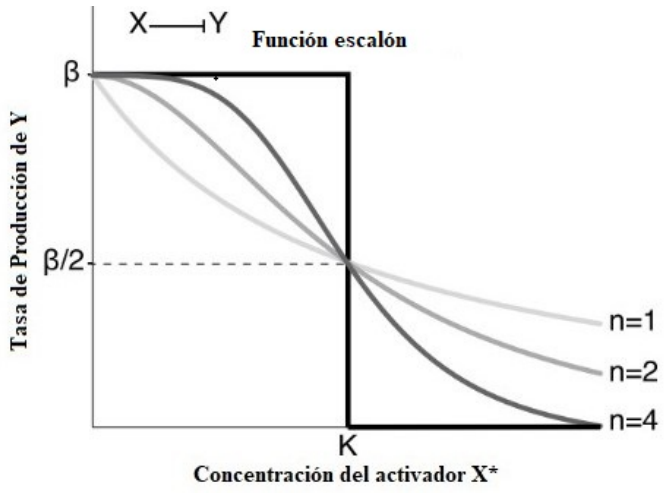


Figura 1.6: Función de Hill para un represor con distintos valores de n.

No obstante, es crucial reconocer que las funciones de Hill presuponen un estado de equilibrio estacionario y, por lo tanto, no reflejan la dinámica temporal de la regulación génica. Los procesos de transcripción y traducción ocurren a lo largo del tiempo, y la velocidad de estos procesos puede influir significativamente en la respuesta de un gen a los cambios en la concentración de un factor de transcripción [34].

1.1.5. Modelo Toggle Switch genético

La aplicación de las funciones de Hill se extiende más allá de la descripción de interacciones individuales, siendo también cruciales en la comprensión y diseño de circuitos genéticos más complejos, como el modelo Toggle Switch.. El circuito genético Toggle Switch es un modelo teórico que representa un sistema de regulación génica con dos genes mutuamente inhibidores. Este circuito se inspira en el interruptor de palanca ('toggle switch' en inglés) que se encuentra en dispositivos electrónicos y se caracteriza por tener dos estados estables, lo que lo convierte en un componente esencial para entender la dinámica de los sistemas biológicos.

En este modelo, dos genes, denominados X y Y, producen proteínas específicas. Estos genes están regulados por promotores y represores que interactúan de manera compleja. La dinámica clave del Toggle Switch radica en que la expresión de un gen inhibe la del otro y viceversa, estableciendo así un mecanismo de conmutación entre dos estados estables (ver Figura 1.7).

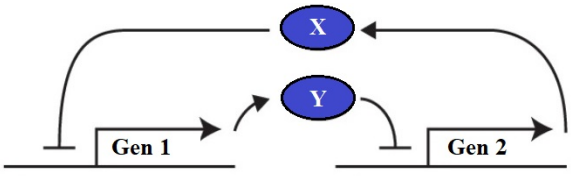


Figura 1.7: Representación del circuito genético Toggle Swich, mostrando la inhibición mutua entre los productos de los genes X y Y.

En el estado 'X activo', el gen X se encuentra activo y produce su proteína correspondiente, X. Esta proteína actúa como un represor para el gen Y. Al unirse al promotor de Y, la proteína X impide la transcripción de Y, manteniéndolo en un estado inactivo. Este mecanismo asegura que, mientras X esté activo y produciendo la proteína X, el gen Y permanecerá reprimido [36].

Por otro lado, el estado 'Y activo' se establece cuando ocurre un cambio en el entorno celular que activa el gen Y. En este escenario, Y comienza a producir la proteína Y, que a su vez inhibe la actividad del gen X. La proteína Y se une al promotor de X, bloqueando su expresión y desactivando la producción de la proteína X. Como resultado, el circuito se estabiliza en el estado 'Y activo' [37].

Este modelo es un ejemplo clásico de un interruptor bi-estable, donde el sistema puede permanecer en uno de los dos estados estables ('X activo' o 'Y activo') hasta que se produce un estímulo externo que provoca la conmutación. La belleza de este sistema radica en su simplicidad y en la capacidad de ilustrar conceptos fundamentales de la regulación génica y la biología sintética [38].

El Toggle Switch no solo es un modelo teórico, sino que también ha sido implementado experimentalmente en organismos como Escherichia coli, demostrando su utilidad práctica en biología sintética. Estos estudios han proporcionado una comprensión más profunda de los mecanismos de conmutación genética y han abierto nuevas vías para el diseño de circuitos genéticos en aplicaciones médicas y biotecnológicas [39].

El comportamiento del Toggle Switch se modela mediante ecuaciones diferenciales que describen las tasas de cambio en las concentraciones de las proteínas X y Y en función del tiempo. Estas ecuaciones permiten estudiar la dinámica del sistema y comprender cómo se producen las transiciones entre los estados estables:

$$\begin{aligned} \frac{dX}{dt} &= \frac{a_1}{1+Y^{n_1}} - d_1 X \\ \frac{dY}{dt} &= \frac{a_2}{1+X^{n_2}} - d_2 Y \end{aligned} \tag{1.4}$$

Donde X es la concentración del represor 1, Y es la concentración del represor 2. Los coeficientes a_1, a_2 y n_1, n_2 son análogos a los que presentamos en la sección anterior, siendo los primeros términos de las ecuaciones funciones de Hill correspondiente a represores. Mientras que el término lineal de cada ecuación representa la degradación/dilución del represor. Este modelo, a pesar de ser sencillo conserva estos dos aspectos fundamentales de una red de regulación genética.

El modelo original del Toggle Switch, propuesto por Gardner, Cantor y Collins en 2000, se centró en la construcción de un interruptor genético en Escherichia coli utilizando dos genes mutuamente inhibidores [36]. Desde entonces, el modelo ha sido ampliado y refinado, con estudios teóricos y experimentales que han explorado la dinámica del sistema, la conmutación entre estados estables y la influencia de diferentes parámetros en el comportamiento del circuito genético.

Variantes y modificaciones del modelo Toggle Switch se han propuesto para adaptarlo a diferentes contextos y aplicaciones, como el modelo de oscilador sintético de Elowitz y Leibler, que utiliza el concepto del Toggle Switch para generar oscilaciones periódicas en las concentraciones de proteínas [37].

Hoy en día, el Toggle Switch sigue siendo un área de investigación activa, explorando aspectos como la influencia del ruido en la conmutación, la robustez del sistema frente a perturbaciones y su interacción con otros componentes de la red de regulación génica.

Las aplicaciones del Toggle Switch se extienden más allá de la biología básica, siendo fundamentales para el diseño y construcción de circuitos genéticos sintéticos en ingeniería genética y biología sintética, con implicaciones en medicina, biotecnología y producción de compuestos industriales.

1.2. Procesos estocásticos

En matemáticas y, en particular, en la teoría de la probabilidad, un proceso estocástico es una colección de variables aleatorias ordenadas, generalmente indexadas por el tiempo. Cada variable aleatoria en este conjunto representa el resultado de un evento o experimento aleatorio. Intuitivamente, un proceso estocástico puede ser visto como una secuencia de eventos aleatorios que evolucionan a lo largo del tiempo, influenciados por probabilidades específicas [40].

Los procesos estocásticos son fundamentales en diversas áreas, desde la física y la biología hasta la economía y la ingeniería. Permiten modelar sistemas en los que el azar juega un papel crucial, como el movimiento de partículas en fluidos, fluctuaciones en mercados financieros o cambios en poblaciones biológicas.

Existen varios tipos de procesos estocásticos, cada uno con características y aplicaciones únicas [41]. Algunos ejemplos notables incluyen:

- **Procesos de Poisson**: Utilizados para modelar eventos que ocurren de manera aleatoria e independiente a lo largo del tiempo.
- **Cadenas de Markov**: Caracterizadas por la propiedad de que el estado futuro depende solo del estado actual.
- **Procesos de renovación**: Se enfocan en tiempos entre eventos sucesivos.
- **Procesos gaussianos**: Son procesos con distribuciones normales multivariadas.
- **Procesos de difusión**: Modelan la evolución continua de variables, como el movimiento browniano.

Formalmente, podemos definir un proceso estocástico como una colección de variables aleatorias $\{X(t) : t \in T\}$ definidas en un espacio de probabilidad $(\Omega, \mathcal{F}, P)$, donde T es un conjunto indexado, a menudo representando tiempo [42].

Cada realización posible de un proceso estocástico corresponde a una función específica, $X(\omega, t) : \Omega \times T \to \mathbb{R}$. Aquí, $\omega \in \Omega$ es un evento particular, y $t \in T$ es un momento en el tiempo. La función $X(\omega, t)$ proporciona el valor del proceso en el tiempo t para el evento ω [41].

Un ejemplo de proceso estocástico es una cadena de Markov, en la cual la probabilidad de transición a cualquier estado futuro depende solo del estado actual y no de cómo se llegó a ese estado. Si denotamos el estado en el tiempo n como X_n, y el estado en el tiempo $n+1$ como X_{n+1}, entonces la propiedad de Markov se puede escribir matemáticamente como:

$$P(X_{n+1} = x_{n+1} | X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) = P(X_{n+1} = x_{n+1} | X_n = x_n) \tag{1.5}$$

donde $P(A|B)$ representa la probabilidad condicional de A dado B [43].

Una generalización de las cadenas de Markov son las ecuaciones diferenciales estocásticas (SDEs, por sus siglas en inglés), que incorporan un término de ruido aleatorio. En términos de biología de sistemas, este ruido puede representar la variabilidad inherente en los procesos bioquímicos a nivel celular [44]. Una SDE típica se puede escribir como:

$$dX_t = a(X_t)dt + b(X_t)dW_t \tag{1.6}$$

donde dX_t es el cambio en el proceso estocástico X_t, $a(X_t)$ es la función de deriva, $b(X_t)$ es la función de difusión, y dW_t es un incremento del movimiento browniano.

Las SDEs son herramientas poderosas para modelar la dinámica de sistemas complejos y pueden resolverse utilizando métodos numéricos como el método de Euler-Maruyama o el método de Milstein, que permiten aproximar soluciones en casos donde las soluciones analíticas no son factibles [45].

1.2.1. Modelo Toggle Switch Genético Estocástico

En un experimento biológico real, rara vez se encuentran resultados precisos y exactos debido a la naturaleza inherente del ruido presente. Este ruido puede provenir de diversas fuentes, como fluctuaciones en las condiciones del experimento, errores en la medición, variabilidad biológica y otros factores no controlados [46]. Por lo tanto, es crucial que cualquier modelo que se construya pueda considerar y manejar este ruido de manera efectiva [47].

En el contexto del Modelo Toggle Switch Genético, proponemos que este ruido proviene primordialmente de la medición experimental [48]. Esto justifica la inclusión de un término de ruido aditivo en las ecuaciones diferenciales que describen el modelo [49].

El sistema se modela de la siguiente manera:

$$\begin{aligned} dX(t) &= \left(\frac{a_1}{1+Y(t)^n} - d_1X(t) + b_1\right)dt + \sigma_1 dW_1(t) \\ dY(t) &= \left(\frac{a_2}{1+X(t)^n} - d_2Y(t) + b_2\right)dt + \sigma_2 dW_2(t) \end{aligned} \tag{1.7}$$

En esta representación:

- $dX(t)$ y $dY(t)$ son los cambios infinitesimales en X y Y respectivamente en el tiempo t.
- σ_1 y σ_2 son los coeficientes que indican la intensidad del ruido aditivo en las ecuaciones para X y Y, respectivamente.
- $dW_1(t)$ y $dW_2(t)$ corresponden a incrementos de un proceso de Wiener (ruido blanco gaussiano) para X y Y, respectivamente . Estos términos simbolizan fluctuaciones aleatorias o ruido, modelados como variables aleatorias con distribución normal.

Cada término de ruido $dW_i(t)$ representa un paso de un camino aleatorio (movimiento browniano), y se entiende que estos incrementos son independientes entre sí para diferentes instantes de tiempo.

1.3. Métodos Numéricos

Los métodos numéricos son técnicas algorítmicas esenciales para realizar aproximaciones numéricas a problemas matemáticos complejos, que a menudo son demasiado intrincados para

resolver mediante métodos analíticos exactos. Estas técnicas son cruciales en una amplia gama de campos científicos y de ingeniería, particularmente en la resolución de ecuaciones diferenciales que modelan fenómenos en física, biología y química.

En la sección anterior, introdujimos el modelo Toggle Switch genético, descrito por un sistema de ecuaciones diferenciales ordinarias. En un escenario ideal, podríamos resolver este sistema de manera analítica para obtener una comprensión detallada de cómo el sistema responde bajo diferentes condiciones. Sin embargo, en la práctica, encontrar soluciones analíticas exactas para sistemas de ecuaciones diferenciales puede ser extremadamente desafiante o incluso inviable.

Aquí es donde las técnicas de resolución numérica cobran una importancia vital. Estos métodos nos permiten abordar y analizar sistemas complejos como el modelo Toggle Switch genético, proporcionando aproximaciones numéricas que nos ayudan a comprender y predecir el comportamiento del sistema en diversas situaciones. A través de la aplicación de métodos numéricos, podemos superar las limitaciones de los enfoques analíticos y explorar en profundidad la dinámica de sistemas biológicos complejos, lo cual es fundamental para avanzar en campos como la biología sintética y la ingeniería genética.

Los métodos numéricos deterministas se utilizan para obtener soluciones aproximadas de ecuaciones o sistemas que no incluyen elementos aleatorios. Estos métodos generan la misma salida cada vez que se ejecutan con un conjunto de entrada dado. Algunos de los métodos deterministas más comunes incluyen:

1. **Método de Euler y Euler Mejorado:** Utilizados para resolver ecuaciones diferenciales ordinarias (EDO), estos métodos aproximan la solución paso a paso a lo largo de un intervalo.

2. **Métodos de Runge-Kutta:** Proporcionan una mayor precisión que el método de Euler y son ampliamente utilizados para resolver EDOs. El método de Runge-Kutta de cuarto orden es particularmente popular debido a su equilibrio entre precisión y eficiencia computacional.

3. **Métodos de Diferencias Finitas:** Empleados para resolver ecuaciones diferenciales parciales (EDP), estos métodos aproximan las derivadas mediante diferencias en valores de puntos de una malla.

A diferencia de los métodos deterministas, los métodos numéricos estocásticos se ocupan de ecuaciones o sistemas que incorporan elementos aleatorios o ruido, como es el caso en muchos modelos biológicos y financieros. Estos métodos son esenciales para simular sistemas que evolucionan con componentes de incertidumbre. Algunos ejemplos incluyen:

1. **Método de Euler-Maruyama:** Una extensión del método de Euler para ecuaciones diferenciales estocásticas (EDE). Es útil para modelar sistemas con ruido aditivo, como en el caso de ciertos modelos biológicos.

2. **Método de Milstein y Métodos de Orden Superior:** Proporcionan una mayor precisión que Euler-Maruyama para EDEs, especialmente cuando el ruido tiene un impacto significativo en el sistema.

3. **Métodos de Monte Carlo:** Utilizados para simular modelos que involucran una gran cantidad de variables aleatorias. Son especialmente útiles en finanzas y física estadística.

1.3.1. Runge-Kutta de quinto orden (RK5)

El método de Runge-Kutta es una técnica numérica ampliamente reconocida para resolver ecuaciones diferenciales ordinarias (EDOs). Se basa en un proceso iterativo que aproxima la

solución de un sistema de ecuaciones a lo largo del tiempo. Entre sus variantes, el método de Runge-Kutta de cuarto orden (RK4) es muy utilizado debido a su equilibrio entre precisión y eficiencia computacional [50].

Para situaciones donde se requiere una precisión aún mayor, se utiliza el método de Runge-Kutta de quinto orden (RK5). Aunque RK5 es computacionalmente más costoso que RK4, su capacidad para proporcionar estimaciones más precisas lo hace valioso en escenarios donde la precisión es crítica y los recursos computacionales son suficientes [51].

El RK5 opera bajo el mismo principio fundamental que el RK4, que es realizar múltiples "predicciones" para el siguiente paso en un intervalo de tiempo y luego combinar estas predicciones para obtener una estimación más precisa. La diferencia clave es que RK5 realiza una predicción adicional y utiliza una fórmula de combinación más compleja para la estimación final.

Supongamos que tenemos una EDO de la forma $\frac{dy}{dt} = f(t, y)$ y queremos calcular la solución y en un intervalo de tiempo desde t_n hasta t_{n+1}. El método RK5 realiza los siguientes pasos:

1. **Calcular los Incrementos Intermedios:** Cada uno de los seis incrementos (k_1 a k_6) se calcula de la siguiente manera:
   - k_1 es el incremento basado en la pendiente al inicio del intervalo.
   - k_2 hasta k_6 son incrementos basados en la pendiente en puntos intermedios del intervalo. Estos puntos intermedios se calculan utilizando una combinación lineal de los incrementos anteriores.

   Por ejemplo, k_2 se calcula utilizando k_1, k_3 se calcula utilizando una combinación de k_1 y k_2, y así sucesivamente

2. **Fórmulas para los incrementos:** Las fórmulas específicas para calcular cada k_i son las siguientes:

$$
\begin{aligned}
k_1 &= hf(t_n, y_n), \\
k_2 &= hf\left(t_n + \frac{1}{4}h, y_n + \frac{1}{4}k_1\right), \\
k_3 &= hf\left(t_n + \frac{3}{8}h, y_n + \frac{3}{32}k_1 + \frac{9}{32}k_2\right), \\
k_4 &= hf\left(t_n + \frac{12}{13}h, y_n + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right), \\
k_5 &= hf\left(t_n + h, y_n + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right), \\
k_6 &= hf\left(t_n + \frac{1}{2}h, y_n - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right),
\end{aligned}
$$

   donde h es el tamaño del paso y f es la función derivada del sistema.

3. **Combinación Ponderada para la Estimación Final:** La nueva estimación de y_{n+1} se calcula utilizando una combinación ponderada de los incrementos:

$$y_{n+1} = y_n + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6$$

Esta combinación ponderada está diseñada para maximizar la precisión de la estimación final.

Implemetación Runge-Kutta de quinto orden para el modelo Toggle Swich

Como ya hemos visto en secciones anteriores, el modelo Toggle Swich en biología de sistemas se puede describir mediante el sistema de ecuaciones diferenciales:

$$\frac{dX}{dt} = \frac{a_1}{1+Y^n} - d_1 X + b_1,$$
$$\frac{dY}{dt} = \frac{a_2}{1+X^n} - d_2 Y + b_2,$$

Para aplicar el método de Runge-Kutta de quinto orden (RK5) a este modelo, seguimos los siguientes pasos:

- Definimos las funciones que representan las tasas de cambio de X y Y. Estas funciones se pueden implementar como funciones separadas o como una sola función que devuelve un vector.
- Escogemos los valores iniciales para X y Y, así como el tamaño del paso de tiempo h.
- En cada paso de tiempo, aplicamos RK5 para calcular los nuevos valores de X y Y. Esto implica calcular los incrementos k_1 a k_6 para cada ecuación y luego combinarlos para obtener la estimación final.
- Repetimos este proceso para cada paso de tiempo hasta alcanzar el final del intervalo de tiempo deseado.

Los incrementos para X se calculan de la siguiente manera:

$$\begin{aligned}
k_{1,X} =& h\left(\frac{a_1}{1+Y^n} - d_1 X + b_1\right),\\
k_{2,X} =& h\left(\frac{a_1}{1+(Y+\frac{1}{4}k_{1,Y})^n} - d_1(X+\frac{1}{4}k_{1,X}) + b_1\right),\\
k_{3,X} =& h\left(\frac{a_1}{1+(Y+\frac{3}{8}k_{1,Y}+\frac{3}{8}k_{2,Y})^n} - d_1(X+\frac{3}{8}k_{1,X}+\frac{3}{8}k_{2,X}) + b_1\right),\\
k_{4,X} =& h\left(\frac{a_1}{1+(Y+\frac{12}{13}k_{1,Y}-\frac{12}{13}k_{2,Y}+\frac{12}{13}k_{3,Y})^n} - d_1(X+\frac{12}{13}k_{1,X}-\frac{12}{13}k_{2,X}+\frac{12}{13}k_{3,X}) + b_1\right),\\
k_{5,X} =& h\left(\frac{a_1}{1+(Y-k_{1,Y}+2k_{2,Y}-k_{3,Y}+k_{4,Y})^n} - d_1(X-k_{1,X}+2k_{2,X}-k_{3,X}+k_{4,X}) + b_1\right),\\
k_{6,X} =& h\Bigg(\frac{a_1}{1+\left(Y+\frac{1}{2}k_{1,Y}-\frac{1}{2}k_{2,Y}+\frac{1}{2}k_{3,Y}-\frac{1}{2}k_{4,Y}+\frac{1}{2}k_{5,Y}\right)^n} - d_1\Bigg(X+\frac{1}{2}k_{1,X}-\frac{1}{2}k_{2,X}+\frac{1}{2}k_{3,X}\\
&-\frac{1}{2}k_{4,X}+\frac{1}{2}k_{5,X}\Bigg) + b_1\Bigg),
\end{aligned}$$

También necesitamos el correspondiente $k_{i,Y}$ del cálculo para Y que se obtienen de manera equivalente.

La aplicación del método RK5 al modelo de toggle switch es un proceso iterativo que requiere cuidado en la implementación. Este método permite simular con precisión la evolución de las concentraciones de las proteínas en el sistema.

1.3.2. Método de Euler-Maruyama para ecuaciones diferenciales estocásticas (SDE)

El método de Euler es un esquema numérico ampliamente utilizado para resolver ecuaciones diferenciales ordinarias, y se ha extendido para su uso en ecuaciones diferenciales estocásticas (SDEs por sus siglas en inglés). El método de Euler para SDEs se conoce a menudo como el método de Euler-Maruyama [52].

Supongamos que tenemos una SDE de la forma:

$$dX_t = a(X_t, t)dt + b(X_t, t)dW_t \tag{1.8}$$

donde a es la componente determinista, b es la función de difusión, y dW_t es un incremento del movimiento browniano.

Para resolver numéricamente esta SDE usando el método de Euler-Maruyama, partimos el intervalo de tiempo de interés en pequeños segmentos de tamaño Δt, y luego iterativamente actualizamos la solución de la siguiente manera:

$$X_{t+\Delta t} = X_t + a(X_t, t)\Delta t + b(X_t, t)\Delta W_t \tag{1.9}$$

donde ΔW_t es una realización de la variable aleatoria normal con media cero y varianza Δt.

El método de Euler-Maruyama es simple y fácil de implementar, pero su precisión depende fuertemente del tamaño del paso Δt. Para SDEs más complicadas o cuando se necesita alta precisión, se pueden requerir métodos numéricos más sofisticados.

Implementación Euler-Maruyama para el modelo Toggle Swich

Aplicando el método Euler-Maruyama a nuestro sistema de ecuaciones 1.4 se tiene:

$$\begin{aligned} X_{i+1} &= X_i + \left(\frac{a_1}{1+Y_i^n} - d1 \cdot X_i + b_1\right) \cdot dt + \sigma_1 \cdot \eta_1 \cdot \sqrt{dt}, \\ Y_{i+1} &= Y_i + \left(\frac{a_2}{1+X_i^n} - d_2 \cdot Y_i + b_2\right) \cdot dt + \sigma_2 \cdot \eta_2 \cdot \sqrt{dt}. \end{aligned} \tag{1.10}$$

Estas ecuaciones reflejan la evolución del sistema bajo la influencia tanto de la dinámica determinista como de las perturbaciones aleatorias que representan el ruido experimental [52].

1.4. Problemas inversos

Los problemas inversos son una rama de la matemática aplicada y la física que se ocupan de la "inversión" de la lógica conocida o supuesta sobre el comportamiento de los sistemas físicos para aprender más sobre los parámetros que rigen ese comportamiento. Estos problemas suelen ser difíciles de resolver y tienen una importancia crucial en diversas áreas de la ciencia y la ingeniería, incluyendo la geofísica, la medicina, la física de partículas, la oceanografía, la astronomía, la acústica, la ecología, y muchas más [53].

En esencia, un problema inverso se ocupa de determinar las propiedades causales de un sistema (el modelo m) a partir de las observaciones de su comportamiento (los datos d), donde se asume una relación física fundamental F entre ambos. Por lo tanto, matemáticamente, podemos expresarlo como:

$$F(m) = d \tag{1.11}$$

En la práctica, las observaciones (datos) casi siempre están influenciadas por el ruido, que puede provenir de una variedad de fuentes, como las limitaciones del instrumento de medición o el redondeo numérico. Así, podemos considerar los datos d como una combinación de las observaciones "reales" sin ruido d_{true} y una componente de ruido η:

$$\begin{aligned} d &= F(m_{true}) + \eta \\ &= d_{true} + \eta \end{aligned} \tag{1.12}$$

Donde d_{true} satisface exactamente la ecuación para $m = m_{true}$. Aquí, un problema directo implica la determinación de los datos d dado un modelo m, mientras que un problema inverso se refiere a la determinación del modelo m dado los datos d.

Cuando el modelo y los datos están representados por un número finito de parámetros, nos referimos a ellos como problemas inversos discretos o problemas de estimación de parámetros [54]. En este caso, los parámetros del modelo se pueden expresar como un vector m de n elementos, y los datos se pueden representar como un vector d de m elementos. El problema de la estimación de parámetros entonces se puede escribir como:

$$F(\mathbf{m}) = \mathbf{d} \tag{1.13}$$

Por otro lado, si tanto el modelo como los datos son funciones de variables continuas (como el tiempo o el espacio), la tarea de estimar m a partir de d se conoce como un problema inverso continuo.

La solución de los problemas inversos a menudo implica técnicas de optimización y/o técnicas estadísticas para tratar con la incertidumbre y el ruido en los datos. Además, los problemas inversos a menudo son mal condicionados, lo que significa que pequeños cambios en los datos pueden llevar a grandes cambios en la solución. Esto puede hacer que la solución de estos problemas sea un desafío.

1.4.1. Problemas inversos en Sistemas de Ecuaciones Diferenciales

En el campo de los problemas inversos aplicados a sistemas de ecuaciones diferenciales, a menudo se busca estimar parámetros desconocidos o funciones que aparecen en las ecuaciones. Esto se realiza utilizando los datos medidos u observados. Por tanto, este tipo de problemas son de interés en una amplia gama de campos, desde la física y la ingeniería hasta la economía y las ciencias biológicas [53].

Uno de los ejemplos clásicos de problemas inversos en sistemas de ecuaciones diferenciales es la identificación de parámetros en los modelos de sistemas dinámicos. En estos casos, las ecuaciones diferenciales describen la dinámica de un sistema y los parámetros representan características del sistema como tasas de reacción, coeficientes de difusión o constantes de tiempo.

El objetivo de un problema inverso en este contexto sería estimar estos parámetros desconocidos a partir de los datos medidos. Esto se hace minimizando alguna función de error que compara las predicciones del modelo (obtenidas resolviendo las ecuaciones diferenciales con un conjunto dado de parámetros) con los datos reales.

Este tipo de problema se puede formular matemáticamente como un problema de optimización:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left(\mathbf{y}_i - \mathbf{f}(t_i, \boldsymbol{\theta})\right)^2, \tag{1.14}$$

donde $\boldsymbol{\theta}$ son los parámetros desconocidos, $\mathbf{y}_i$ son los datos medidos, $\mathbf{f}(t_i, \boldsymbol{\theta})$ son las predicciones del modelo en el tiempo t_i y el sumatorio se realiza sobre todas las mediciones disponibles.

Para resolver este tipo de problemas, se pueden utilizar varios métodos numéricos de optimización, como el método de Newton, el algoritmo de Levenberg-Marquardt o diversos métodos de optimización global.

Es importante mencionar que los problemas inversos son notoriamente difíciles de resolver, en parte debido a su naturaleza mal condicionada. Esto significa que pequeñas variaciones en los datos pueden dar lugar a grandes variaciones en la solución. Por esta razón, es común emplear técnicas de regularización para estabilizar el problema [55].

Para el caso de mínimos cuadrados, la función objetivo se convertiría en la suma de los residuos cuadrados entre las observaciones y las simulaciones del sistema de ecuaciones, evaluado en los puntos de observación y sumado para todas las observaciones:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i} \left[\left(X_{obs,i} - \frac{a_1}{1 + Y_i^n} + d_1 X_i + b_1 \right)^2 + \left(Y_{obs,i} - \frac{a_2}{1 + X_i^n} + d_2 Y_i + b_2 \right)^2 \right] \quad (1.15)$$

1.4.2. Breve Introducción a las Redes Neuronales en la Resolución de Problemas Inversos

Las redes neuronales, una rama fundamental del aprendizaje automático y la inteligencia artificial, han emergido como herramientas poderosas para abordar una amplia gama de problemas complejos. Estos sistemas computacionales, inspirados en la estructura neuronal del cerebro humano, son capaces de aprender patrones y relaciones subyacentes en grandes conjuntos de datos.

En el contexto de problemas inversos, donde el objetivo es deducir causas desconocidas a partir de efectos observados, las redes neuronales ofrecen un enfoque prometedor. Estos problemas, comunes en campos como la física, la ingeniería y las ciencias biomédicas, a menudo implican descifrar parámetros ocultos o estados de un sistema a partir de datos medidos o experimentales.

Las redes neuronales son particularmente adecuadas para problemas inversos debido a su capacidad para modelar relaciones complejas y no lineales. A través del entrenamiento con datos, una red neuronal puede aprender a mapear observaciones a los parámetros o estados subyacentes del sistema en estudio. Esta capacidad las hace herramientas valiosas para:

1. **Análisis de Datos Complejos:** Pueden manejar y aprender de grandes volúmenes de datos, incluso cuando las relaciones entre los datos de entrada y salida son intrincadas y no lineales.

2. **Manejo de Incertidumbre y Ruido:** Las redes neuronales son robustas frente a datos ruidosos o incompletos, lo que es común en mediciones reales.

3. **Flexibilidad y Adaptabilidad:** Pueden adaptarse a una variedad de problemas y tipos de datos, lo que las hace aplicables en múltiples disciplinas.

La integración de redes neuronales en la resolución de problemas inversos representa un avance significativo en la capacidad de analizar y comprender sistemas complejos. Al aprender de los datos, estas redes pueden proporcionar insights valiosos y soluciones a problemas que tradicionalmente han sido desafiantes de resolver. Su uso en problemas inversos es un ejemplo claro de cómo la inteligencia artificial está abriendo nuevas fronteras en la investigación científica y tecnológica.

1.5. Redes Neuronales

1.5.1. Introducción

La inteligencia artificial (IA) es un campo interdisciplinario que se centra en el desarrollo de sistemas y programas informáticos capaces de realizar tareas que normalmente requieren de la inteligencia humana. Estos sistemas están diseñados para simular la capacidad de razonamiento, aprendizaje y toma de decisiones de los seres humanos [56].

Dentro de la IA, el Machine Learning (aprendizaje automático) es una rama clave que se enfoca en desarrollar algoritmos y modelos que permiten a las máquinas aprender de datos y realizar tareas sin ser programadas explícitamente. El Machine Learning utiliza técnicas estadísticas y matemáticas para identificar patrones y generar modelos predictivos a partir de grandes conjuntos de datos [57].

Uno de los enfoques más poderosos dentro del Machine Learning es el Deep Learning (aprendizaje profundo). Se basa en redes neuronales artificiales con múltiples capas ocultas que imitan la estructura y función del cerebro humano. Estas redes profundas son capaces de aprender representaciones de alto nivel de los datos, lo que les permite realizar tareas complejas como reconocimiento de imágenes, procesamiento de lenguaje natural y conducción autónoma [58]. El éxito del Deep Learning se debe en gran medida a los avances en el poder de cálculo y el acceso a grandes conjuntos de datos. La disponibilidad de hardware especializado, como las unidades de procesamiento gráfico (GPU), ha acelerado el entrenamiento y la inferencia de modelos de Deep Learning.

La IA, el Machine Learning y el Deep Learning tienen aplicaciones en una amplia gama de sectores, incluyendo la medicina, la industria automotriz, la atención al cliente, la seguridad cibernética y muchos más. Estas tecnologías están impulsando avances significativos en la automatización de tareas, la toma de decisiones inteligentes y la generación de conocimiento a partir de datos.

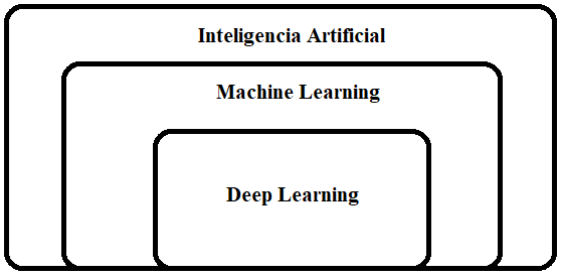


Figura 1.8: Jerarquía de la Inteligencia Artificial. Deep Learning es un subconjunto de Machine Learning que a su vez es un subconjunto de la inteligencia artificial.

1.5.2. Machine Learning

El aprendizaje automático, o machine learning en inglés, es una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos que permiten a las computadoras aprender y mejorar automáticamente a partir de datos [56]. A diferencia de la programación tradicional, en la que se le indican a la máquina las reglas y los pasos a seguir, el aprendizaje automático se basa en la capacidad de la máquina para aprender patrones y tomar decisiones por sí misma.

El concepto de aprendizaje automático ha existido desde hace décadas, pero ha ganado una gran relevancia en los últimos años gracias al aumento en la capacidad de procesamiento, el acceso a grandes volúmenes de datos y los avances en algoritmos. El objetivo principal del aprendizaje automático es desarrollar modelos y algoritmos que puedan generalizar a partir de ejemplos o experiencias pasadas y aplicar ese conocimiento para hacer predicciones o tomar decisiones en situaciones nuevas.

Técnicas de Machine Learning se aplican en una amplia variedad de campos y sectores, como la medicina, la industria, las finanzas, la publicidad, la seguridad, entre otros. Se utiliza para resolver problemas complejos, como el reconocimiento de imágenes, el procesamiento del lenguaje natural, la detección de fraudes, la recomendación de productos y muchas otras aplicaciones.

Dependiendo de las necesidades del problema, el ambiente en el que se van a desenvolver y los factores que afectarán la toma de decisiones, podemos encontrar distintos tipos de algoritmos de aprendizaje, entre los cuales vamos a hablar de 2 de ellos: supervisado y no supervisado.

Aprendizaje Supervisado

El aprendizaje supervisado es un enfoque del aprendizaje automático en el que se entrenan modelos utilizando ejemplos etiquetados. En este tipo de aprendizaje, se proporcionan al modelo tanto las características de entrada como las salidas esperadas correspondientes. El objetivo del modelo es aprender a mapear las entradas a las salidas de manera que pueda realizar predicciones precisas en datos no vistos.

El proceso de aprendizaje supervisado implica la construcción de un modelo basado en un conjunto de datos de entrenamiento que contiene ejemplos etiquetados. Estos ejemplos consisten en pares de entradas y salidas conocidas, donde las salidas representan las respuestas deseadas o las etiquetas para las entradas dadas. Durante la fase de entrenamiento, el modelo ajusta sus parámetros internos para minimizar la diferencia entre las salidas predichas y las salidas reales conocidas.

Una vez que el modelo ha sido entrenado, se evalúa su rendimiento utilizando un conjunto de datos separado llamado conjunto de prueba o conjunto de validación. Este conjunto de datos contiene ejemplos no vistos por el modelo durante el entrenamiento y se utiliza para medir su capacidad de generalización. El rendimiento del modelo se puede evaluar utilizando métricas como la precisión, la exactitud, el valor F o el error cuadrático medio, dependiendo del tipo de problema y las salidas esperadas.

Existen diversos algoritmos utilizados en el aprendizaje supervisado, como los clasificadores lineales, los árboles de decisión, las máquinas de vectores de soporte (SVM), las redes neuronales y muchos otros [58]. La elección del algoritmo adecuado depende del tipo de problema, la naturaleza de los datos y los requisitos de rendimiento.

Aprendizaje no supervisado

El aprendizaje no supervisado es un enfoque del aprendizaje automático en el que los modelos se entrenan sin utilizar ejemplos etiquetados. A diferencia del aprendizaje supervisado, en el que se proporcionan salidas esperadas, en el aprendizaje no supervisado, el modelo debe descubrir patrones y estructuras en los datos por sí mismo.

En el aprendizaje no supervisado, se exploran los datos en busca de similitudes, agrupaciones o relaciones ocultas. Los algoritmos de aprendizaje no supervisado pueden realizar tareas como la

clusterización, que consiste en agrupar datos similares en conjuntos, o la reducción de dimensionalidad, que busca representar los datos en un espacio de menor dimensión.

El aprendizaje no supervisado se aplica en diversas áreas, como la exploración de datos, la detección de anomalías, la segmentación de imágenes, la recomendación de productos y muchas más [58]. Los algoritmos de aprendizaje no supervisado, como el algoritmo k-means, el análisis de componentes principales (PCA) y las redes neuronales autoencoder, son ampliamente utilizados en estas aplicaciones.

Aunque el aprendizaje no supervisado no tiene la guía explícita de salidas etiquetadas, puede ayudar a descubrir patrones interesantes y generar información útil para la toma de decisiones. Es un campo activo de investigación en el aprendizaje automático y continúa evolucionando para abordar desafíos en el análisis y comprensión de datos.

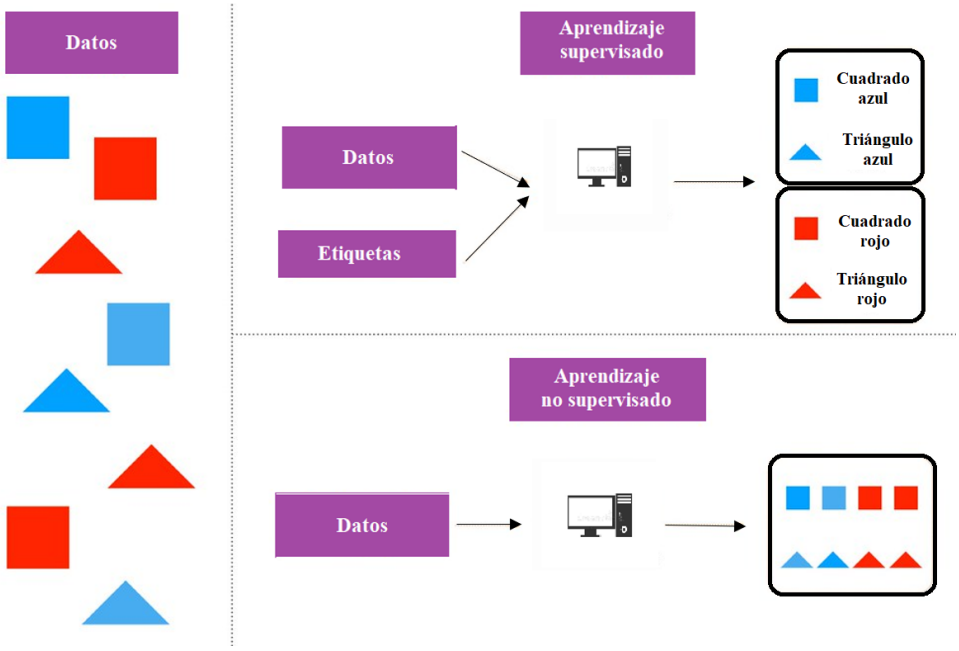


Figura 1.9: Comparación entre aprendizaje supervisado y no supervisado: clasificación con etiquetas frente a agrupación por características.

1.5.3. Algoritmo de Redes neuronales artificiales

Un caso especial de algoritmos de Machine Learning son las redes neuronales artificiales. Se suelen utilizar para problemas de Clasificación y Regresión pero realmente tienen un gran potencial para resolver multitud de problemáticas debido a que son muy buenas para detectar patrones.

Están inspiradas en el funcionamiento del cerebro humano y su red de neuronas [59]. Estas redes están compuestas por unidades interconectadas llamadas neuronas artificiales o nodos, que trabajan en conjunto para procesar información y realizar tareas específicas.

La neurona artificial, elemento fundamental de estas redes, es un modelo computacional basado en las neuronas biológicas [60]. Las neuronas naturales reciben señales a través de sinapsis

ubicadas en las dendritas o membrana de la neurona [61]. Cuando las señales recibidas son lo suficientemente fuertes (superan un determinado umbral), la neurona se activa y emite una señal [62]. Esta señal podría enviarse a otra sinapsis y podría activar otras neuronas.

El entrenamiento de una red neuronal artificial implica ajustar los pesos y las conexiones entre las neuronas para que la red pueda aprender a realizar una tarea específica. Esto se logra mediante el uso de algoritmos de aprendizaje, como el algoritmo de retropropagación [63], que calcula y ajusta los gradientes de los pesos en función del error de salida.

Las redes neuronales artificiales han demostrado ser herramientas potentes para afrontar diversos problemas de aprendizaje automático, incluyendo el reconocimiento de patrones, clasificación de imágenes, procesamiento de lenguaje natural y predicción de series temporales. Su habilidad para aprender y adaptarse a partir de datos las convierte en modelos flexibles y eficientes para capturar relaciones complejas en los datos.

Con el avance de la tecnología informática y el aumento en la disponibilidad de grandes conjuntos de datos, las redes neuronales artificiales han experimentado un resurgimiento en las últimas décadas. Además, el desarrollo de arquitecturas más profundas, como las redes neuronales profundas o deep learning [58], ha permitido el entrenamiento de modelos aún más complejos y sofisticados.

La estructura básica de una red neuronal artificial incluye capas de neuronas interconectadas organizadas jerárquicamente. Esta estructura generalmente comprende una capa de entrada, una o varias capas ocultas y una capa de salida [59]. La capa de entrada es la primera capa de la red y se encarga de recibir los datos de entrada. Cada neurona en esta capa representa una característica o atributo de los datos y transmite la información a las neuronas de la capa oculta. Las capas ocultas son las capas intermedias entre la capa de entrada y la capa de salida. Estas capas están compuestas por neuronas que procesan la información recibida de las capas anteriores y generan salidas que se transmiten a las capas posteriores. El número y la configuración de las capas ocultas pueden variar dependiendo de la complejidad del problema y la arquitectura de la red neuronal. Finalmente, la capa de salida es la última capa de la red y produce los resultados o predicciones finales. Cada neurona en esta capa representa una clase o valor de salida y genera una salida correspondiente.

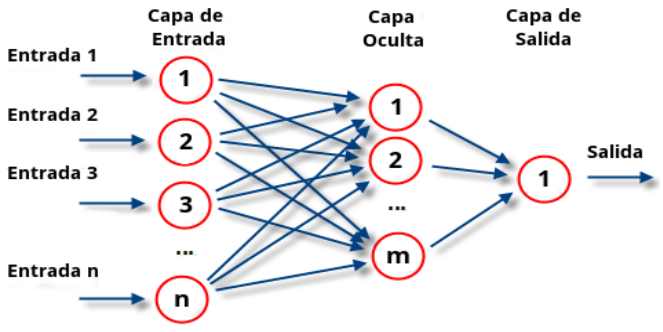


Figura 1.10: Estructura de una red neuronal artificial mostrando la interconexión entre la capa de entrada, las capas ocultas y la capa de salida.

La Figura 1.10 ilustra una red neuronal artificial conformada por tres capas esenciales: una capa de entrada, una capa oculta, y una capa de salida. En esta representación, las neuronas se simbolizan

mediante círculos. La capa de entrada de n neuronas, situada al principio de la red, se encarga de recibir los datos iniciales. Seguidamente, la capa oculta, compuesta por m neuronas, procesa estos datos. Cada neurona en esta capa tiene la capacidad de realizar operaciones de cálculo y activación. Las conexiones entre las neuronas de diferentes capas permiten la transmisión y transformación de información a través de la red. Finalmente, la capa de salida recoge las señales procesadas de la capa oculta y genera la predicción final o el resultado del proceso. Es importante destacar que las conexiones entre las neuronas de distintas capas son fundamentales para el funcionamiento de la red, permitiendo la propagación y modificación de la información a lo largo de la estructura.

1.5.4. La unidad básica de procesamiento de una red neuronal artificial: el perceptrón

El análogo a las neuronas naturales se le conoce como perceptrón. Los perceptrones fueron desarrollados en las décadas de 1950 y 1960 por el científico Frank Rosenblatt, inspirado en trabajos anteriores de Warren McCulloch y Walter Pitts. Un perceptrón toma varios valores de entrada binarios $x_1, x_2, \ldots$, y produce una sola salida binaria como en la Figura 1.11 [64].

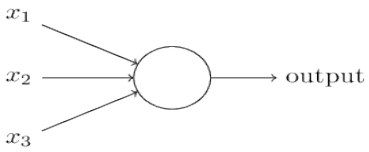


Figura 1.11: Arquitectura de un perceptrón con tres entradas x_1, x_2, x_3

Rosenblatt propuso una regla simple para calcular la salida, introdujo pesos $w_1, w_2, \ldots$, siendo números reales que expresaban la importancia de la respectiva entrada para la salida. La salida de la neurona, 0 ó 1, es determinada si la suma ponderada $\sum_j w_j x_j$ es mayor o menor que algún valor umbral. Al igual que los pesos, el umbral es un número real que es un parámetro de la neurona [65]. Para decirlo en términos algebraicos más precisos:

$$salida = \begin{cases} 0 & \text{si } \sum_j w_j x_j \leq \text{valor umbral} \\ 1 & \text{si } \sum_j w_j x_j > \text{ valor umbral} \end{cases} \tag{1.16}$$

Ese es el modelo matemático básico. Una forma de pensar en el perceptrón es que es un dispositivo que toma decisiones dependiendo de la información que se le otorgue. Un ejemplo simple y fácil de entender es el siguiente:

Supongamos que se acerca el fin de semana y hemos oído que habrá un festival de videojuegos en la ciudad. Nos gustan los videojuegos y estamos intentando decidir si vamos o no al festival. Podemos tomar su decisión debido a tres factores:

1. *¿El clima esta agradable?*
2. *¿Hay alguien que pueda acompañarnos?*
3. *¿El festival está cerca del transporte público?*

Podemos representar estos tres factores mediante las correspondientes variables binarias x_1, x_2 y x_3. Por ejemplo, tendríamos $x_1 = 1$ si el clima es bueno y $x_1 = 0$ si el clima es malo. Del mismo modo, $x_2 = 1$ si tenemos compañía, y $x_2 = 0$ si no. Y lo mismo de nuevo para x_3 y el transporte público.

Ahora, supongamos que adoramos los videojuegos, tanto que iríamos al festival incluso si no tenemos quien nos acompañe y es difícil llegar al festival. Pero quizás realmente detestamos el mal tiempo, y no hay forma de que vayamos al festival si hace mal tiempo. Podemos utilizar perceptrones para modelar este tipo de toma de decisiones. Una forma de hacer esto es elegir un peso $w_1 = 6$ *para el clima, y* $w_2 = 2$ *y* $w_3 = 2$ *para las otras condiciones. El valor más alto de* w_1 *indica que el clima nos importa mucho, mucho más que si alguien nos acmompaña, o la cercanía del transporte público. Finalmente, elegimos un umbral de 5 para el perceptrón. Con estas opciones, el perceptrón implementa el modelo de toma de decisiones deseado, generando 1 cuando el clima es bueno y 0 cuando el clima es malo. No importa en la salida de la compañía o si el transporte público está cerca.*

Variando los pesos y el umbral, podemos obtener diferentes modelos de toma de decisiones. Por ejemplo, supongamos que en su lugar elegimos un umbral de 3. Entonces el perceptrón decidiría que debes ir al festival siempre que haga buen tiempo o cuando el festival esté cerca del transporte público y tu alguien esté dispuesto a acompañarlo. En otras palabras, sería un modelo diferente de toma de decisiones. Bajar el umbral significa que estás más dispuesto a ir al festival.

El perceptrón no representa un modelo integral de la toma de decisiones humana. Sin embargo, el ejemplo sirve para ilustrar cómo este puede evaluar y considerar diferentes tipos de evidencia al tomar decisiones.

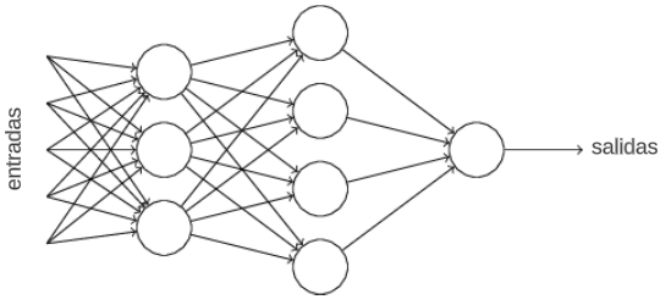


Figura 1.12: Modelo de una pequeña red de perceptrones compleja.

En la red de la Figura 1.12, la primera columna de perceptrones, lo que llamaremos la primera capa de perceptrones, está tomando tres decisiones muy simples, que deriva de la evidencia de entrada. Cada uno de los perceptrones de la segunda capa toma una decisión tomando como valores de entrada los resultados de la primera capa de toma de decisiones. De esta manera, un perceptrón de la segunda capa puede tomar una decisión a un nivel más complejo y abstracto que los perceptrones de la primera capa. Y el perceptrón de la tercera capa puede tomar decisiones aún más complejas. De esta manera, una red de perceptrones de muchas capas puede participar en una toma de decisiones sofisticada.

Aunque en la Figura 1.12 los perceptrones parecen tener múltiples salidas, al final, sigue siendo una única salida. Las flechas de salida múltiple son simplemente una forma útil de indicar que la salida de un perceptrón se utiliza como entrada a todos los perceptrones de la siguiente capa.

En la Ecuación 1.16 podemos escribir $\sum_j w_j x_j$ como un producto escalar, $w \cdot x \equiv \sum_j w_j x_j$, donde w y x son vectores cuyos componentes son los pesos y las entradas, respectivamente. También se puede mover el umbral al otro lado de la desigualdad y reemplazarlo por lo que se conoce como el sesgo del perceptrón, $b \equiv$ umbral. Usando el sesgo en lugar del umbral, la regla del perceptrón se puede reescribir como:

$$salida = \begin{cases} 0 & \text{si } w \cdot x + b \leq 0 \\ 1 & \text{si } w \cdot x + b > 0 \end{cases} \tag{1.17}$$

Resulta que podemos idear algoritmos de aprendizaje que puedan ajustar automáticamente los pesos y sesgos de una red de neuronas artificiales. Esta sintonía ocurre en respuesta a estímulos externos, sin la intervención directa de un programador. Estos algoritmos de aprendizaje nos permiten usar neuronas artificiales de una manera radicalmente diferente a las puertas lógicas convencionales.

Supongamos que tenemos una red de perceptrones que nos gustaría usar para aprender a resolver algún problema. Por ejemplo, las entradas a la red pueden ser los datos de píxeles sin procesar de una imagen escaneada y manuscrita de un dígito. Y nos gustaría que la red aprendiera los pesos y los sesgos para que la salida de la red clasifique correctamente el dígito. Para ver cómo podría funcionar el aprendizaje, suponga que hacemos un pequeño cambio en algún peso (o sesgo) en la red. Lo que nos gustaría es que este pequeño cambio de peso provoque solo un pequeño cambio correspondiente en la salida de la red. Lo que buscamos se puede representar en la Figura 1.13.

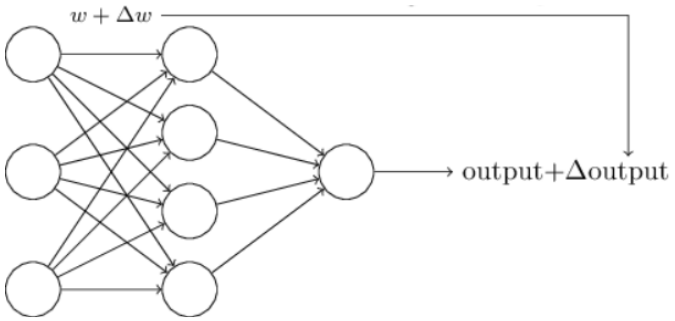


Figura 1.13: Visualización de cómo un pequeño cambio en los pesos (Δ w) puede afectar la salida de una red de perceptrones, ilustrando el desafío de ajustar los pesos para lograr una clasificación precisa sin alterar significativamente el comportamiento general de la red.

Si fuera cierto que un pequeño cambio en un peso (o sesgo) causa solo un pequeño cambio en la salida, entonces podríamos usar este hecho para modificar los pesos y sesgos para que nuestra red se comporte más de la manera que queremos. Por ejemplo, supongamos que la red clasifica por error una imagen como un "8" cuando debería ser un "9". Podríamos averiguar cómo hacer un pequeño cambio en los pesos y sesgos para que la red se acerque un poco más a clasificar la imagen como un "9". Y luego repetíamos esto, cambiando los pesos y los sesgos una y otra vez para producir un resultado cada vez mejor. La red estaría aprendiendo.

El problema es que esto no es lo que sucede cuando nuestra red contiene perceptrones. Un pequeño cambio en los pesos o el sesgo de cualquier perceptrón individual en la red a veces puede hacer que la salida de ese perceptrón cambie completamente, digamos de 0 a 1. Ese cambio puede hacer que el comportamiento del resto de la red cambie completamente de una manera muy complicada. Entonces, aunque su "9" ahora podría estar clasificado correctamente, es probable que el comportamiento de la red en todas las demás imágenes haya cambiado por completo de alguna manera difícil de controlar.

Podemos superar este problema introduciendo un nuevo tipo de neurona artificial llamada **neurona sigmoide**. Las neuronas sigmoides son similares a los perceptrones, pero modificadas

de modo que pequeños cambios en sus pesos y sesgos provocan solo un pequeño cambio en su salida. Ese es el hecho crucial que permitirá que una red de neuronas sigmoides aprenda.

Tienen entradas $x_1, x_2, \ldots$, pero en lugar de que éstas sean solo valores 0 y 1, pueden ser todos los valores que van de 0 a 1. Tal como el perceptrón las neuronas sigmoides tienen pesos para cada entrada $w_1, w_2, \ldots$ y un sesgo general b; sin embargo la salida no solo son valores 0 y 1, en vez de ello son de la forma $\sigma(w \cdot x + b)$ donde σ es llamada la función sigmoide definida como:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1.18}$$

Que de una manera más explícita las salidas de una neurona sigmoide con entradas $x_1, x_2, \ldots$, pesos $w_1, w_2, \ldots$, y sesgo b es:

$$\frac{1}{1 + exp\left(-\sum_j w_j x_j - b\right)} \tag{1.19}$$

La forma de la función sigmoide se puede ver en la Figura 1.16.

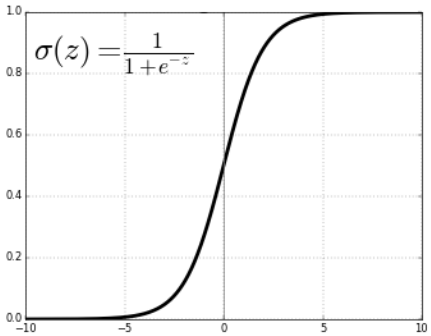


Figura 1.14: Forma de la función sigmoide

Esta forma es una versión suavizada de una función escalonada que es precisamente la forma de un perceptrón:

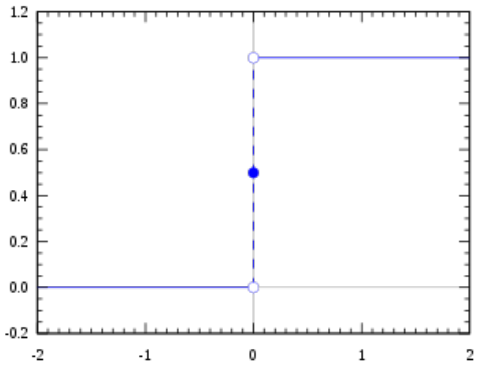


Figura 1.15: Forma de la función que define a un perceptrón

El hecho crucial es la suavidad de la función σ, no su forma detallada. La suavidad de σ significa que pequeños cambios Δw_j en los pesos y Δb en el sesgo producirán un pequeño cambio Δsalida

en la salida de la neurona. De hecho, el cálculo nos dice que Δ salida está bien aproximada por:

$$\Delta salida \approx \sum_j \frac{\partial salida}{\partial w_j}\Delta w_j + \frac{\partial salida}{\partial b}\Delta b$$

Donde la suma es sobre todos los pesos w_j y $\partial salida/\partial w_j$ y $\partial salida/\partial b$ denotan derivadas parciales de la salida con respecto a w_j y b respectivamente.

Si bien la expresión anterior parece complicada, con todas las derivadas parciales, en realidad dice algo muy simple: $\Delta salida$ es una función lineal de los cambios Δw_j y Δb en los pesos y el sesgo. Esta linealidad facilita la elección de pequeños cambios en los pesos y sesgos para lograr cualquier pequeño cambio deseado en la salida. Entonces, si bien las neuronas sigmoides tienen gran parte del mismo comportamiento cualitativo que los perceptrones, hacen que sea mucho más fácil descubrir cómo cambiar los pesos y los sesgos cambiará la salida.

La neurona sigmoide es solo una **función de activación**. Sin embargo existen neuronas donde la salida es $f(w \cdot x + b)$ para alguna otra función $f(\cdot)$. Lo principal que cambia cuando usamos una función de activación diferente es que los valores particulares de las derivadas parciales.

1.5.5. Funciones de activación

Las funciones de activación en las redes neuronales son fundamentales para que estas puedan modelar relaciones no lineales entre las entradas y las salidas. De hecho, si no se utilizan funciones de activación no lineales, una red neuronal profunda no sería más poderosa que una red de una sola capa, independientemente de cuántas capas tenga, ya que la composición de una serie de transformaciones lineales es todavía una transformación lineal [58].

Algunas de las funciones de activación más comunes incluyen la función sigmoide, la tangente hiperbólica (tanh), la unidad lineal rectificada (ReLU), entre otras.

1. **Sigmoide**: La función sigmoide es una función que toma cualquier valor de entrada y lo comprime en un rango entre 0 y 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.20}$$

   Donde x es la entrada a la neurona. A pesar de su utilidad inicial en el campo de las redes neuronales artificiales, la función sigmoide presenta inconvenientes técnicos cuando se manejan entradas con valores absolutos grandes. En tales casos, las derivadas de la función se vuelven insignificantes, un fenómeno conocido como **desvanecimiento del gradiente**. Este problema dificulta la actualización eficaz de los pesos durante el entrenamiento de la red mediante algoritmos basados en el gradiente, como la retropropagación, especialmente en redes con muchas capas. Por este motivo, aunque la función sigmoide sigue siendo relevante en ciertos contextos, se han desarrollado y adoptado otras funciones de activación que mitigan este problema.

2. **Tangente Hiperbólica (tanh)**: La función tanhes una función de activación en redes neuronales que, al igual que la función sigmoide, comprime los valores de entrada, pero lo hace en un rango que va de -1 a 1. Esto la diferencia de la sigmoide, que los comprime en el intervalo de 0 a 1. La fórmula matemática para la función tanh es la siguiente:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.21}$$

La función tanh ofrece una transición suave y diferenciable que es simétrica alrededor del origen, lo cual puede ser ventajoso en ciertas aplicaciones de aprendizaje automático al centrar la salida de la función. Al igual que la función sigmoide, la función tanh puede experimentar el problema del desvanecimiento de gradientes durante el entrenamiento de la red, especialmente cuando se trata con valores de entrada que tienen magnitudes grandes. Esto ocurre porque las derivadas de la función tanh tienden a valores muy pequeños a medida que las entradas se alejan de cero, lo que puede ralentizar o incluso detener el aprendizaje durante la fase de retropropagación, ya que las actualizaciones de los pesos se vuelven insignificantes.

3. **Unidad Lineal Rectificada (ReLU)**: La función ReLU es actualmente la función de activación más utilizada en el entrenamiento de redes neuronales profundas, principalmente debido a su eficiencia computacional y a que mitiga (aunque no elimina completamente) el problema del gradiente que se desvanece [66]. La definición matemática de la ReLU es:

$$\text{ReLU}(x) = \text{máx}(0, x) \tag{1.22}$$

Esta función asigna un valor de cero a todas las entradas negativas y mantiene sin cambios las entradas positivas. La linealidad de la ReLU para valores positivos permite un cálculo eficiente del gradiente y acelera la convergencia durante el entrenamiento usando métodos basados en gradientes.

No obstante, la ReLU no está exenta de desafíos; uno notable es el problema de las "neuronas muertas". Una neurona se considera "muerta" si se activa siempre con un valor de cero, sin importar la entrada recibida. Esto puede suceder si un gradiente grande actualiza los pesos de tal manera que la activación resultante siempre es negativa; estas neuronas entonces dejan de contribuir en el proceso de aprendizaje.

Para mitigar este inconveniente, se han propuesto variantes de ReLU como la Leaky ReLU y la ELU (Exponential LU). Estas variantes introducen una pequeña pendiente positiva para entradas negativas, lo que permite que las neuronas "muerta" puedan eventualmente reactivarse durante el entrenamiento, ya que permiten una pequeña derivada positiva cuando $x < 0$, favoreciendo la continuidad del aprendizaje incluso en presencia de valores negativos en la entrada.

Las funciones de activación modernas, como la sigmoide, tanh y ReLU, han suplantado la función de activación de umbral binario empleada en el perceptrón clásico, el cual solo emitía respuestas binarias de 0 o 1. La limitante principal de la función de umbral es su indiferenciabilidad, lo cual impide el uso de métodos de optimización basados en gradientes, tales como el descenso por gradiente, necesarios para el entrenamiento efectivo de redes neuronales.

A diferencia de la función de umbral, las funciones de activación contemporáneas son diferenciables en casi todo su dominio (o en su totalidad, como en el caso de la sigmoide y tanh). La ReLU, aunque no es diferenciable en el punto cero, se considera "casi diferenciable" y práctica para el uso en algoritmos de aprendizaje. Estas características diferenciables son fundamentales para la retropropagación, la técnica prevalente para el ajuste de pesos en redes neuronales profundas [58]. Este avance metodológico permite a las redes neuronales aprender de una manera más eficiente y efectiva, facilitando el entrenamiento en capas múltiples y el manejo de datos complejos.

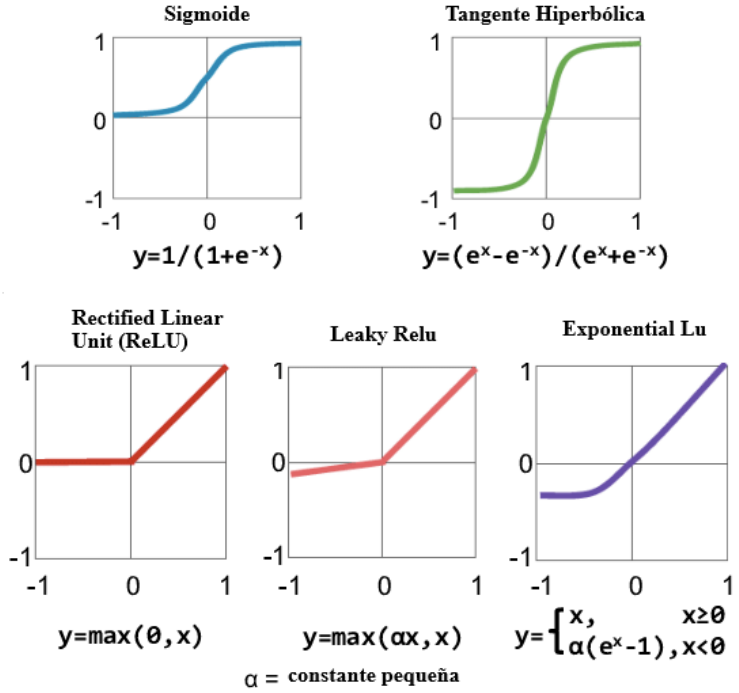


Figura 1.16: Comparación de Funciones de Activación en Redes Neuronales: Sigmoide, Tangente Hiperbólica, ReLU, Leaky ReLU y Exponential Lu.

1.5.6. Funciones de Costo

Las funciones de costo, también referidas como funciones de pérdida, son una pieza angular en el entrenamiento de las redes neuronales. Proporcionan una medida esencial que refleja la efectividad con la que la red neuronal está desempeñando la tarea para la que ha sido entrenada.

En el núcleo del proceso de entrenamiento de una red neuronal yace un ciclo iterativo de ajuste fino. Durante este ciclo, los pesos y sesgos de la red se calibran meticulosamente para minimizar la función de costo. Este ajuste se realiza frecuentemente mediante algoritmos de optimización basados en gradientes, como el descenso por gradiente y sus variantes evolucionadas, que se benefician de la naturaleza diferenciable de estas funciones de costo.

La selección de la función de costo es una decisión estratégica que varía en función del tipo de tarea de aprendizaje automático que se está abordando. Entre las funciones de costo más prevalentes se encuentran:

1. **Error Cuadrático Medio (MSE)**: Predilecta para tareas de regresión, el MSE cuantifica el promedio de los errores al cuadrado entre las predicciones y los valores reales. Esta función es sensible a los errores más grandes debido a la elevación al cuadrado, lo que a menudo conduce a un modelo más robusto contra las desviaciones grandes.

$$L_{\text{MSE}}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{1.23}$$

   Donde y son los valores verdaderos, $\hat{y}$ son las predicciones de la red y N es el número de ejemplos [58].

2. **Mean Absolute Error (Mean Absolute Error MAE)**: Utilizada en tareas de regresión, el MAE es una medida más directa de los errores, calculando el promedio de las diferencias absolutas entre las predicciones y los valores verdaderos. Este método asigna un peso uniforme a todos los errores, lo que puede ser beneficioso en presencia de valores atípicos.

$$L_{\text{MAE}}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{1.24}$$

   Donde y son los valores verdaderos, $\hat{y}$ son las predicciones de la red y N es el número de ejemplos.

3. **Pérdida de Entropía Cruzada (Cross-Entropy Loss)**: Es la función de costo preferida para problemas de clasificación, comparando la distribución de probabilidad predicha por la red con la distribución de probabilidad real de los datos. Esta función penaliza las predicciones incorrectas con una alta eficacia, siendo especialmente útil cuando las salidas de la red están normalizadas para formar una distribución de probabilidad.

$$L_{\text{CE}}(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{1.25}$$

   Donde y son los valores verdaderos codificados como one-hot vectors y $\hat{y}$ son las predicciones de la red [58]. La entropía cruzada penaliza las predicciones que están lejos de los valores verdaderos y es especialmente eficaz cuando la salida de la red se normaliza para formar una distribución de probabilidad (como con la función softmax).

Además de guiar el entrenamiento, las funciones de costo son herramientas valiosas para la evaluación del modelo. Comparando los valores de la función de costo obtenidos en los conjuntos de entrenamiento y validación, es posible determinar si la red está generalizando correctamente o si está sobreajustando los datos de entrenamiento.

Es importante reconocer que existen múltiples funciones de costo diseñadas para aplicaciones específicas, como la función de log-verosimilitud negativa para modelos generativos o la función de costo de margen para máquinas de vectores de soporte. La selección adecuada de la función de costo depende intrínsecamente del problema específico a abordar y de la naturaleza de los datos involucrados [73].

1.5.7. Optimizadores

En el campo del aprendizaje automático y, en particular, en el entrenamiento de redes neuronales, los optimizadores juegan un papel crucial. Estos algoritmos se encargan de ajustar los parámetros del modelo, como los pesos y los sesgos, con el objetivo principal de minimizar la función de costo. Entre los diversos algoritmos de optimización, el Descenso de Gradiente destaca por ser el fundamento sobre el que se construyen muchas variantes, cada una adaptada a necesidades específicas y desafíos de optimización.

Algoritmo del Descenso de Gradiente

Central en el repertorio de optimizadores en Machine Learning, el Descenso de Gradiente es un método clásico y ampliamente adoptado para minimizar eficientemente la función de costo en una variedad de aplicaciones, incluyendo las redes neuronales [58]. Este algoritmo se basa en un enfoque iterativo y sistemático para ajustar los parámetros del modelo, guiándose por el gradiente

de la función de costo para encontrar la dirección de la mejora más rápida.

El concepto de Descenso de Gradiente se basa en la observación de que si una función multivariable $f(\boldsymbol{x})$ es definida y diferenciable en el punto $\boldsymbol{a}$, entonces $f(\boldsymbol{x})$ decrece más rápidamente si uno va desde $\boldsymbol{a}$ en la dirección del gradiente negativo de f en $\boldsymbol{a}$, $-\nabla f(\boldsymbol{a})$ [58].

Aquí, $\nabla f(\boldsymbol{a})$ denota el vector gradiente de la función f en el punto $\boldsymbol{a}$, y $-\nabla f(\boldsymbol{a})$ es simplemente este vector multiplicado por -1, dando la dirección de máxima pendiente descendente. En términos de una red neuronal, $\boldsymbol{x}$ serían los parámetros (pesos y bias) de la red, y $f(\boldsymbol{x})$ sería la función de costo que queremos minimizar.

El algoritmo del Descenso de Gradiente actualiza los parámetros $\boldsymbol{x}$ iterativamente de la siguiente manera:

$$\boldsymbol{x}_{\text{nuevo}} = \boldsymbol{x}_{\text{anterior}} - \eta \nabla f(\boldsymbol{x}_{\text{anterior}}) \tag{1.26}$$

donde η es la tasa de aprendizaje, un hiperparámetro que controla cuánto cambiamos los parámetros en cada actualización [58].

Es importante destacar que la tasa de aprendizaje debe ser configurada correctamente. Si es demasiado grande, el algoritmo puede diverger y no converger a un mínimo. Si es demasiado pequeña, el algoritmo tardará mucho tiempo en converger. Algunas estrategias, como el descenso de gradiente con tasa de aprendizaje adaptativa, ajustan la tasa de aprendizaje durante el entrenamiento para tratar de obtener las ventajas de una tasa de aprendizaje alta (convergencia rápida) y baja (convergencia estable). Una forma intuitiva de entender el Algoritmo del Descenso de Gradiente se puede ver en el Apéndice A.

Descenso de Gradiente Estocástico (SGD)

Una adaptación importante del Descenso de Gradiente es el Descenso de Gradiente Estocástico (SGD). A diferencia del método tradicional, que utiliza todo el conjunto de datos para calcular el gradiente en cada iteración, el SGD actualiza los parámetros utilizando solo una muestra o un pequeño lote de ejemplos por vez. Esta estrategia mejora la eficiencia computacional y acelera la convergencia, especialmente en grandes conjuntos de datos.

El SGD ofrece ventajas clave sobre el Descenso de Gradiente por lote completo:

- Eficiencia Computacional: Utilizar subconjuntos de los datos en cada iteración reduce significativamente la carga computacional.
- Convergencia más rápida: Las actualizaciones frecuentes pueden conducir a una convergencia más rápida, aunque cada actualización individual sea menos precisa.
- Potencial para Escapar de Mínimos Locales: La aleatoriedad en la selección de muestras ayuda al SGD a escapar de mínimos locales en la superficie de error.

Existen numerosos algoritmos avanzados que han sido desarrollados para abordar diferentes desafíos y mejorar la eficacia del proceso de entrenamiento. Estos incluyen el SGD con Momento, que agrega un componente de inercia a las actualizaciones para una convergencia más rápida, Adagrad, que adapta individualmente la tasa de aprendizaje a cada parámetro, RMSProp, una mejora de Adagrad que previene su rápida disminución de la tasa de aprendizaje, y Adam, que combina las ventajas de Momentum y RMSProp para optimizaciones más eficientes. Estos algoritmos demuestran la evolución continua en la búsqueda de optimizadores más efectivos, cada uno con características únicas que pueden ser más adecuadas para ciertos tipos de problemas de aprendizaje automático.

Descenso de Gradiente Estocástico con Momento (Momentum SGD)

Este es una modificación del Descenso de Gradiente Estocástico que tiene en cuenta la dirección de los gradientes anteriores para suavizar las actualizaciones de los pesos. Esto se logra introduciendo un término de momento que actúa como una "inercia", lo que significa que una vez que los pesos comienzan a moverse en una dirección, seguirán haciéndolo en esa dirección [68]. Esto puede ayudar a acelerar la convergencia, especialmente en problemas con superficies de error planas o cuando hay ruido en los gradientes. Matemáticamente, el SGD con momento se puede expresar de la siguiente manera:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta), \\ \theta &= \theta - v_t, \end{aligned} \tag{1.27}$$

donde v_t es la velocidad en el tiempo t, γ es el parámetro de momento (un valor típico sería 0.9), η es la tasa de aprendizaje, $\nabla_\theta J(\theta)$ es el gradiente de la función de pérdida J con respecto a los parámetros θ, y θ son los parámetros que estamos tratando de optimizar.

Adagrad (Adaptive Gradient Algorithm)

Adagrad es un algoritmo de optimización que adapta la tasa de aprendizaje a los parámetros. Realiza actualizaciones de menor magnitud para los parámetros asociados con características frecuentes y actualizaciones de mayor magnitud para los parámetros asociados con características infrecuentes. Es por lo tanto muy útil para datos dispersos [69]. El Adagrad se puede representar como sigue:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot \nabla_\theta J(\theta_{t,i}), \tag{1.28}$$

donde $G_{t,ii}$ es la suma de los cuadrados de los gradientes pasados respecto a θ_i y ϵ es un número muy pequeño para prevenir la división por cero (normalmente alrededor de $1e-8$).

RMSProp (Root Mean Square Propagation)

RMSProp es una mejora del Adagrad que resuelve su problema de monótonamente decreciente de la tasa de aprendizaje. En lugar de acumular todas las gradientes pasadas, RMSProp solo mantiene un promedio móvil de los cuadrados de las gradientes [70]. El algoritmo RMSProp puede ser representado matemáticamente de la siguiente forma:

$$\begin{aligned} E[g^2]_t &= 0{,}9 E[g^2]_{t-1} + 0{,}1 g_t^2, \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t, \end{aligned} \tag{1.29}$$

Adam (Adaptive Moment Estimation)

Adam combina las ventajas de Momentum SGD y RMSProp. Al igual que Momentum, mantiene un promedio móvil de las gradientes pasadas, y al igual que RMSProp, mantiene un promedio móvil de los cuadrados de las gradientes. Adam ha demostrado ser eficaz en la práctica y es actualmente uno de los algoritmos de optimización más recomendados en el entrenamiento de redes neuronales

[71]. El algoritmo Adam se puede representar de la siguiente manera:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \end{aligned} \tag{1.30}$$

donde m_t y v_t son estimaciones del primer momento (la media) y el segundo momento (la varianza no centrada) de los gradientes respectivamente.

Estos son solo algunos de los algoritmos de optimización disponibles para entrenar redes neuronales y cada uno tiene sus propias ventajas y desventajas. La elección del algoritmo de optimización puede tener un impacto significativo en la velocidad de convergencia y en la calidad final de la red entrenada. Es común experimentar con distintos optimizadores y ajustar sus hiperparámetros para obtener el mejor desempeño posible en una tarea específica de aprendizaje automático.

Estos optimizadores son cruciales para afinar los parámetros del modelo de manera efectiva, conduciendo al objetivo de minimizar la función de costo. Sin embargo, para que estos ajustes sean posibles, primero debemos comprender cómo se calculan los gradientes que estos optimizadores utilizan. Aquí es donde entra en juego el algoritmo de retropropagación, una piedra angular en el entrenamiento de redes neuronales. La retropropagación es el mecanismo por el cual se determinan los gradientes de la función de costo con respecto a los parámetros del modelo, permitiendo a los optimizadores como el SGD y sus variantes realizar actualizaciones informadas y dirigidas. La siguiente sección se sumerge en los detalles de cómo funciona la retropropagación, estableciendo la conexión entre la evaluación del desempeño del modelo y la mecánica de su mejora continua a través del aprendizaje.

1.5.8. Algoritmo de backpropagation

El algoritmo de **Backpropagation**, también conocido como propagación hacia atrás, es un método utilizado en redes neuronales para calcular la derivada del error en función de los pesos de la red [72]. Esta derivada, o gradiente, se utiliza luego en un algoritmo de optimización, como el descenso de gradiente, para ajustar los pesos de la red con el fin de minimizar el error.

El principio de la retropropagación reside en la regla de la cadena del cálculo diferencial. En el contexto de una red neuronal, que se puede considerar como una composición de varias funciones (capas), la derivada de la función de salida respecto a cualquier peso interno se calcula multiplicando las derivadas sucesivas de cada función intermedia. Este proceso consta de dos fases principales: la propagación hacia adelante y la propagación hacia atrás.

En la **propagación hacia adelante**, las entradas se pasan a través de la red, capa por capa, hasta que se alcanza la capa de salida y se produce una predicción. Durante este proceso, se almacenan los valores intermedios que serán necesarios durante la propagación hacia atrás.

Una vez que se ha producido una predicción, se calcula el error comparando la predicción con el valor real. Este error se propaga entonces hacia atrás a través de la red en el paso de **propagación hacia atrás**.

Matemáticamente, si tenemos una red neuronal con una sola capa oculta, los pesos se actualizan de la siguiente manera durante la retropropagación:

1. Para los pesos de la capa de salida:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta(t_i - o_i)o_i(1 - o_i)h_j,$$

   donde η es la tasa de aprendizaje, E es el error, t_i es el valor objetivo, o_i es la salida de la red, y h_j es la salida de la capa oculta.

2. Para los pesos de la capa oculta:

$$\Delta v_{jk} = -\eta \frac{\partial E}{\partial v_{jk}} = -\eta \sum_i (t_i - o_i)o_i(1 - o_i)w_{ij}h_j(1 - h_j)x_k,$$

   donde x_k es la entrada de la red, y los demás símbolos son como se definió anteriormente.

Este proceso se repite, actualizando los pesos después de cada paso hacia atrás, hasta que el error de la red se minimiza a un nivel aceptable o hasta que se alcanza un número máximo de iteraciones. Cabe mencionar que la retropropagación asume que todas las funciones en la red son diferenciables, y el algoritmo puede no funcionar como se espera si este no es el caso [73].

Un ejemplo numérico de cómo se aplica el Algoritmo de Backpropagation se puede ver en el Apéndice B.

1.5.9. Redes convolucionales

El proceso dentro de una red densa es el siguiente: se recibe un vector como entrada y se multiplica con una matriz para producir una salida (a la que generalmente se agrega un vector de sesgo antes de pasar el resultado a través de una no linealidad). Esto es aplicable a cualquier tipo de entrada, ya sea una imagen, un clip de sonido o una colección desordenada de características, sin importar su dimensionalidad, su representación siempre se puede aplanar en un vector antes de la transformación. Sin embargo, las imágenes, los clips de sonidos y otros datos similares tienen una relación intrínseca estructurada. Esta estructura no se explota cuando se aplica una transformación normal dentro de una red neuronal. De hecho, todos los ejes se tratan de la misma manera y no se tiene en cuenta la información topológica [74]. Aún así, aprovechar la estructura implícita de los datos puede resultar muy útil para resolver algunas tareas. Aquí es donde entran en escena las redes convolucionales.

Una convolución discreta es una transformación lineal que preserva la noción del orden [75]. La Figura 1.17 se ilustra un ejemplo de una convolución simple. La cuadrícula azul clara se denomina **mapa de características de entrada**. En este ejemplo se ilustra una sola capa de características de entrada, pero pueden estar apiadas una encima de la otra. Un **kernel** (área sombreada) de valor:

0	1	2
2	2	0
0	1	2

se desliza a través del mapa de características de entrada. En cada ubicación se calcula el producto entre cada elemento del kernel y el elemento de entrada al que se superpone y los

resultados se suman para obtener la salida en la ubicación actual [76].

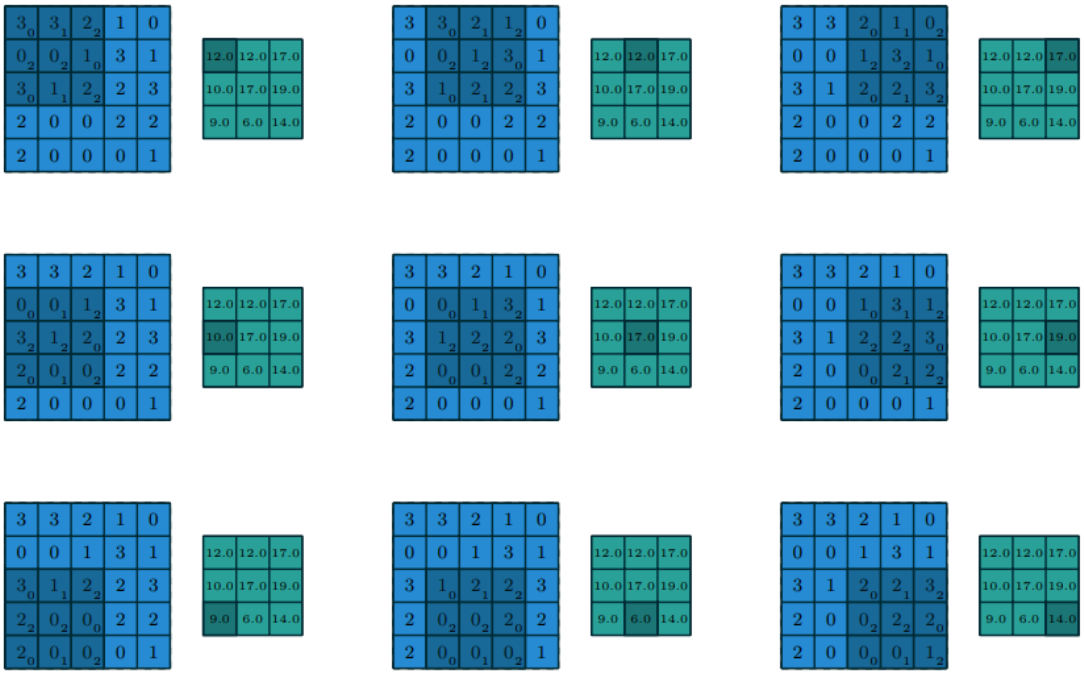


Figura 1.17: Visualización de múltiples mapas de características resultantes de aplicar filtros en una capa convolucional 2-D de una red neuronal.

Los resultados de este procedimiento se denominan **mapas de características de salida**. La convolución mostrada en la Figura 1.17 es un ejemplo de una convolución 2-D, pero se puede generalizar a convolucionales N-D. En una convolución tridimensional, el kernel sería un paralelepípedo y se deslizaría a lo largo de la altura, el ancho y la profundidad del mapa de características de entrada [77].

El conjunto de kernels que define una convolución discreta tiene una forma que corresponde a alguna permutación de $(n, m, k_1, \cdots, k_N)$, donde n es el número de mapas de características de salida, m es el número de mapas de características de entrada y k_j es el tamaño del núcelo a lo largo del eje j.

Capas de agrupación (pooling)

Como se mencionó antes, las operaciones de agrupación reducen el tamaño de los mapas de características mediante el uso de alguna función para resumir las subregiones, como tomar el promedio o el valor máximo.

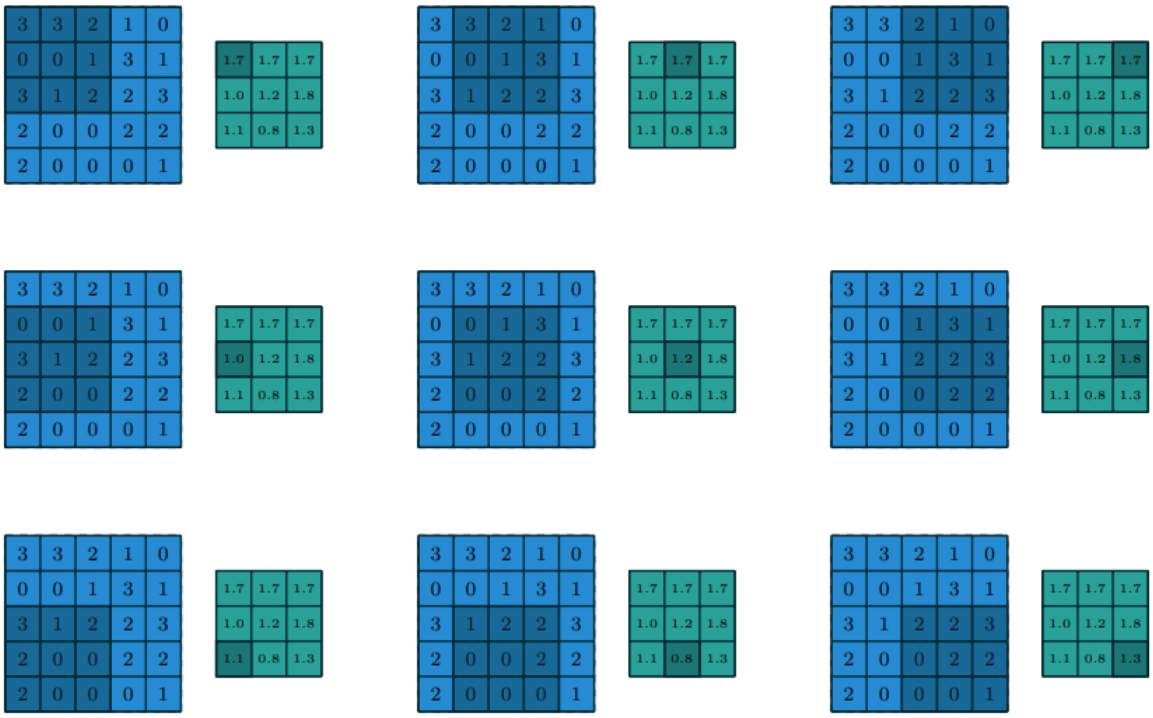


Figura 1.18: Cálculos de una operación average pooling con un kernel de 3×3.

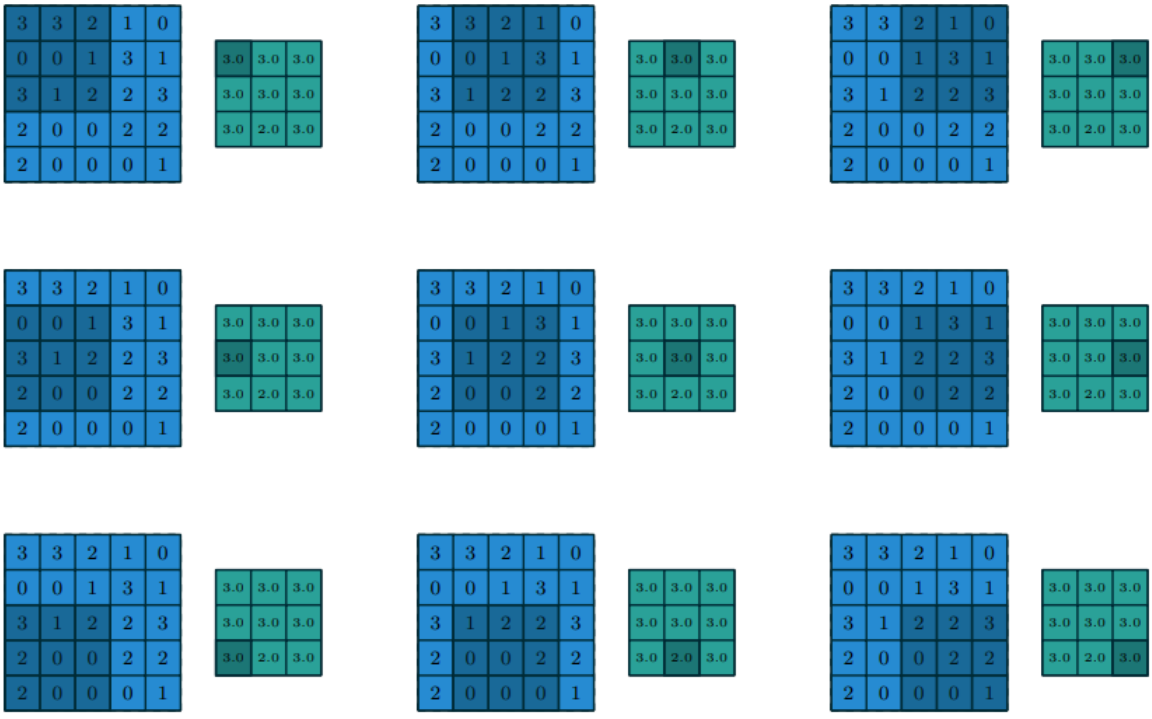


Figura 1.19: Cálculos de una operación max pooling con un kernel de 3×3.

Las capas de pooling funcionan deslizando una ventana a través de la entrada y alimentando el contenido de la ventana a una función de agrupación. En cierto sentido, el pooling funciona como una capa convolucional, pero reemplaza la combinación lineal descrita por el núcleo con alguna otra función. La Figura 1.18 proporciona un ejemplo de "average pooling' ' (agrupación promedio), mientras que la Figura 1.19 se puede ver un ejemplo de "max pooling' ' (agrupación máxima).

Es interesante remarcar que con la transformación de pooling mantenemos la relación espacial. Para verlo visualmente, en la Figura 1.20 es una matriz de 28×28 donde se representa un número 3 en escala de grises. Si aplicamos una operación de max pooling con una ventana de 2×2 obtenemos una matriz de 14×14 donde se mantiene una representación que nos recuerda sin duda alguna a un número 3.

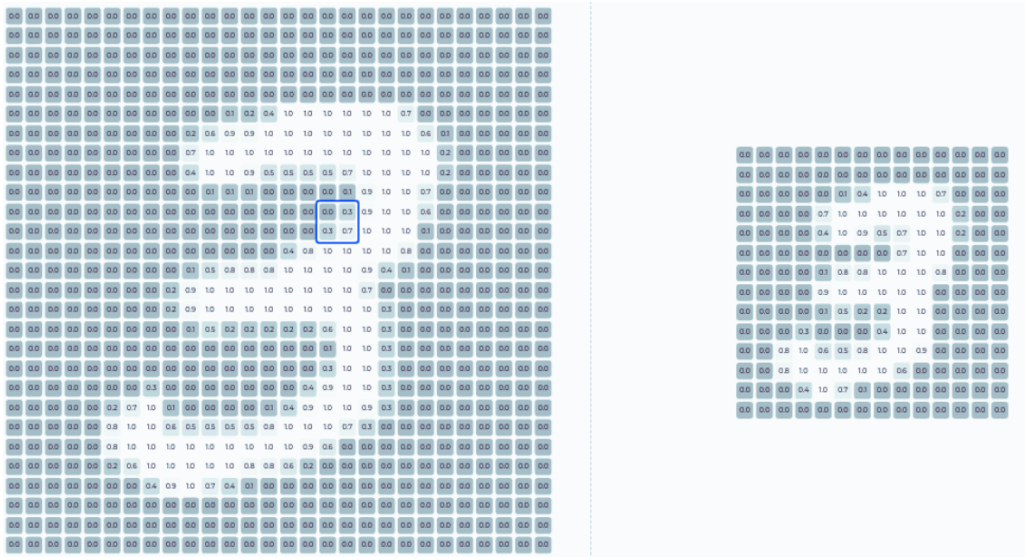


Figura 1.20: Proceso de pooling en redes neuronales convolucionales, comparando la reducción de dimensionalidad antes y después de aplicar la operación.

Aritmética de las redes neuronales convolucionales

El caso más simple de analizar es cuando el kernel simplemente se desliza por todas las posiciones de entrada (es decir, $s = 1$ y $p = 0$). La Figura 1.21 nos da un ejemplo para $i = 4$ y $k = 3$.

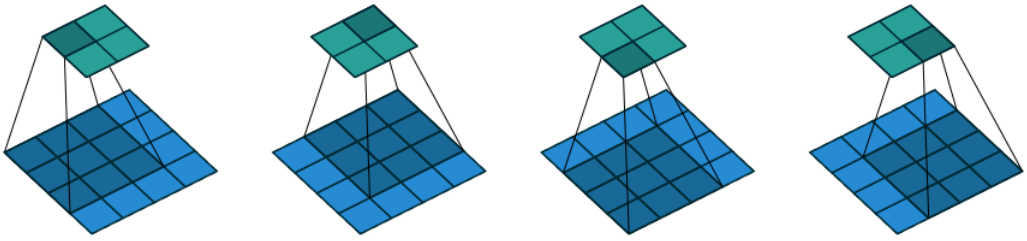


Figura 1.21: Convolución sin padding, stride = 1.

La forma de definir el tamaño de salida en este caso es por el número de ubicaciones posibles del kernel en la entrada. Consideremos el eje de ancho: el kernel comienza en la parte más a la izquierda del mapa de características de entrada y se desliza en pasos de uno hasta que toca el lado derecho de la entrada. El tamaño de la salida será igual al número de pasos realizados más uno, teniendo en cuenta la posición inicial del kernel (Figura 1.22).

Por lo tanto podemos llegar que para $s = 1$ y $p = 0$, tenemos que la salida:

$$o = (i - k) + 1 \tag{1.31}$$

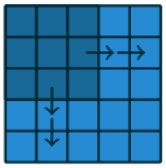


Figura 1.22: Proceso de stride en la operación de convolución. El kernel tiene que deslizarse dos pasos hacia la derecha para tocar el lado derecho de la entrada y de manera equivalente hacia abajo. Agregando uno para tener en cuenta la posición inicial del kernel, tenemos que la salida es de 3×3

Para el caso en que se agreguen dimensiones a la capa de entrada mediante el uso de padding, tenemos que la capa de entrada cambia de i a $i + 2p$, siendo p el número de dimensiones de ceros que se agrega a una dimensión. Por lo que con este cambio y la Ecuación 1.31 tenemos que para cualquier i, k y p para $s = 1$:

$$o = (i - k) + 2p + 1 \tag{1.32}$$

En la Figura 1.23 se puede ver un ejemplo para $i = 5$, $k = 4$ y $p = 2$. Sustituyendo en la Ecuación 1.32, tenemos:

$$o = (5 - 4) + (2 \times 2) + 1 = 6$$

Que es precisamente lo que se puede ver en la Figura 1.23.

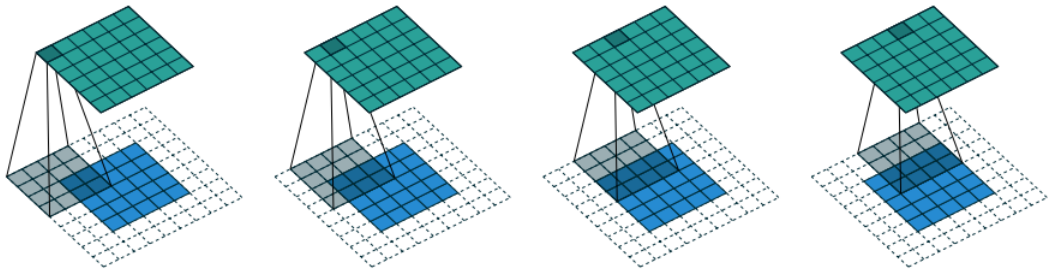


Figura 1.23: Ejemplo con padding, donde $p = 1$ y $s = 1$.

Si deseamos que la salida sea del mismo tamaño que la entrada ($o = i$), tenemos a partir de la

Ecuación 1.32

$$\begin{aligned} o &= (i-k)+2p+1 = i \\ \Rightarrow i &= (i-k)+2p+1 \\ \Rightarrow 0 &= -k+2p+1 \\ \therefore p &= \frac{k-1}{2} \end{aligned} \tag{1.33}$$

Esto se denomina como same padding o half padding. La Figura 1.24 indica un kernel de 3×3 que recorre una entrada de 5×5 con un padding de ceros de 2×2 usando pasos unitarios (es decir i=5, k=3, s=1 y p=1). Podemos ver que cumple con la condición de la Ecuación 1.33 por lo que la salida es del mismo tamaño que la entrada.

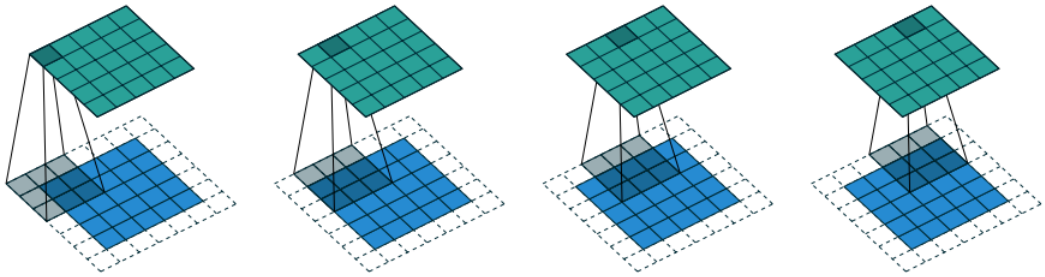


Figura 1.24: En este ejemplo, la salida de la capa es igual a la entrada de la capa, ya que cumple con la Ecuación 1.33 (i=5, k=3, s=1 y p=1). A esto se le denomina con same padding.

Todo lo anterior fue hecho para un stride igual a 1. Para el caso en que el stride sea diferente de cero y no haya pooling ($s > 1$ y $p = 0$). El tamaño de la salida se puede definir en términos de la cantidad de ubicaciones posibles del kernel en la entrada. La Figura 1.25 proporciona un ejemplo para $i = 5$, $k = 3$ y $s = 2$.

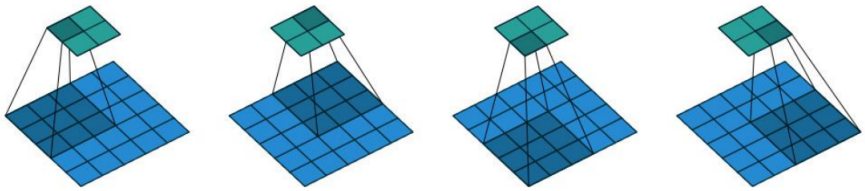


Figura 1.25: Secuencia de operaciones de convolución con un stride mayor a uno, ilustrando cómo el filtro se mueve en intervalos superiores a una unidad sobre la matriz de entrada, resultando en una reducción del tamaño espacial de la matriz de salida en comparación con un stride de uno.

Considerando el eje del ancho, vemos que el kernel comienza como siempre en la parte más a la izquierda de la entrada, pero esta vez se desliza en pasos de tamaño s hasta que toca el lado derecho de la entrada. El tamaño de la salida es nuevamente igual al número de pasos realizados, más uno. La misma lógica se aplica para cada uno de los ejes. Por lo que para cuando $p = 0$:

$$o = \left\lfloor \frac{i-k}{s} \right\rfloor + 1 \tag{1.34}$$

La función piso en la Ecuación 1.35 explica el hecho de que a veces el último paso posible no coincide con el núcleo que llega al final de la entrada, como se puede ver en la Figura 1.26.

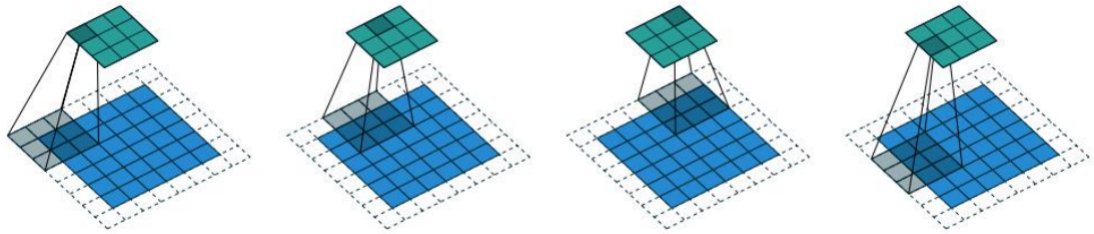


Figura 1.26: Ejemplo de convolución con un stride de $s = 1$ y $p = 1$, demostrando cómo el tamaño de la salida se calcula utilizando la Ecuación 1.35 y cómo el último paso del kernel no siempre coincide con el borde de la entrada.

El caso más general ($s > 0$ y $p > 0$) se puede derivar utilizando la Ecuación 1.35 con una entrada igual a $i + 2p$, es decir:

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1 \tag{1.35}$$

1.6. Software para el aprendizaje automático

1.6.1. Introducción

En el ámbito del aprendizaje automático, existen diferentes lenguajes de programación y bibliotecas de software que facilitan la implementación y aplicación de algoritmos de aprendizaje automático. Entre estos, Python se ha destacado como uno de los lenguajes de programación más utilizados, debido a su simplicidad, flexibilidad y la gran cantidad de bibliotecas de software disponibles que permiten implementar de manera eficiente distintos algoritmos de aprendizaje automático [80].

1.6.2. Python: Lenguaje de Programación

Python es un lenguaje de programación interpretado de alto nivel que se diseñó con un enfoque en la legibilidad del código. En comparación con otros lenguajes como $C++$ o Java, Python permite a los desarrolladores escribir programas funcionales con menos líneas de código. Esto se logra gracias a una sintaxis simple y directa y al manejo automático de la memoria. Además, Python es un lenguaje de propósito general que se puede utilizar para una amplia gama de aplicaciones, desde el desarrollo web hasta la ciencia de datos [80].

Es un lenguaje de programación interpretado, interactivo y orientado a objetos, creado por Guido van Rossum en la década de 1980 [81]. Python es un lenguaje de alto nivel que proporciona estructuras de datos potentes junto con una forma de escribir programas clara y eficiente. Desde sus inicios, ha estado diseñado con la legibilidad del código en mente, con un fuerte énfasis en la simplicidad y la brevedad del código fuente.

Sus principales características son su simplicidad y la legibilidad del código, que se logran en gran parte a través del uso de una sintaxis clara y de la indentación del código como parte de la sintaxis del lenguaje. Esto es diferente a muchos otros lenguajes de programación, donde la indentación se utiliza sólo como una cuestión de estilo y no tiene significado para el lenguaje en

sí. En Python, la indentación es obligatoria y se utiliza para delimitar bloques de código. Esto hace que el código sea más fácil de leer y entender, ya que la estructura del código se refleja en su apariencia visual [82].

Python también se ha ganado el favor de la comunidad científica y de la inteligencia artificial debido a su fácil acceso a las bibliotecas eficientes para la manipulación de datos y la realización de cálculos numéricos. Entre estas bibliotecas se incluyen NumPy [83] para el manejo de grandes matrices de datos y el cálculo numérico, SciPy [84] para el cálculo científico, y Pandas [85] para la manipulación y análisis de datos.

Además, Python ha sido adoptado por la comunidad de aprendizaje automático y redes neuronales debido a la disponibilidad de bibliotecas poderosas como TensorFlow [86], Keras y Scikit-Learn [87].

1.6.3. Keras y TensorFlow: Bibliotecas para Redes Neuronales

TensorFlow es una biblioteca de software de código abierto para el aprendizaje automático y las redes neuronales desarrollada por Google Brain Team. Desde su lanzamiento en 2015, TensorFlow ha tenido un impacto significativo en el campo del aprendizaje automático y la inteligencia artificial debido a su potente funcionalidad y flexibilidad.

En su núcleo, TensorFlow es un sistema para realizar cálculos numéricos con diagramas de flujo de datos, donde los nodos en el gráfico representan operaciones matemáticas, y los bordes representan los datos multidimensionales (tensores) que fluyen entre las operaciones. Esta estructura de gráfico permite a TensorFlow manejar eficientemente grandes conjuntos de datos y realizar cálculos complejos en múltiples CPU o GPU. Esto lo hace particularmente útil para aplicaciones de aprendizaje automático y redes neuronales, donde el manejo eficiente de grandes conjuntos de datos y la capacidad de realizar cálculos paralelos es esencial.

Keras, por otro lado, es una interfaz de programación de aplicaciones (API) de alto nivel para redes neuronales construida sobre TensorFlow. Keras se creó con el objetivo de permitir una rápida experimentación con redes neuronales y proporcionar una forma más fácil y simplificada de construir y entrenar redes neuronales en comparación con las herramientas de bajo nivel proporcionadas por TensorFlow .

Keras proporciona una serie de características clave que facilitan la creación de redes neuronales, incluyendo la capacidad de definir fácilmente capas de red, funciones de activación y optimizadores, una variedad de tipos de capas predefinidos que pueden ser utilizados para construir una red neuronal (por ejemplo, capas densas, capas de convolución, capas recurrentes), y soporte para numerosas técnicas de regularización y optimización [87].

El Apéndice C presenta ejemplos detallados de implementación de redes neuronales densas y convolucionales utilizando Keras y TensorFlow. Esta sección demuestra la facilidad y flexibilidad que ofrecen estas herramientas para definir y entrenar modelos de redes neuronales. La simplicidad y eficacia de TensorFlow y Keras han sido factores clave en su amplia adopción dentro de la comunidad de aprendizaje automático, como se discute en [86].

1.6.4. Optimización de Hiperparámetros con Optuna

Optuna es una biblioteca de software para la optimización de hiperparámetros en modelos de aprendizaje automático, crucial en aprendizaje profundo donde la selección adecuada de hiperparámetros impacta significativamente el rendimiento del modelo [88]. Automatiza la

búsqueda de la mejor combinación de hiperparámetros, mejorando la eficiencia y efectividad de los modelos. Utiliza técnicas de búsqueda y optimización para explorar el espacio de hiperparámetros, basándose en pruebas sucesivas que guían las búsquedas futuras [88]. Soporta varios algoritmos de optimización, incluyendo la optimización bayesiana y el descenso de gradiente.

En este estudio, Optuna se integró para optimizar los hiperparámetros de las redes neuronales utilizadas. El objetivo era encontrar la configuración óptima que maximizara la precisión de las predicciones de los parámetros del modelo Toggle Switch. Las variables de hiperparámetros incluían la tasa de aprendizaje, la cantidad y tamaño de las capas ocultas, las funciones de activación, y otros aspectos relevantes del modelo de red neuronal.

Capítulo 2

Metodología

2.1. Introducción

Esta investigación se centra en un objetivo fundamental: determinar los parámetros de un sistema de ecuaciones diferenciales que describe una red de regulación genética conocida como Toggle Switch. La hipótesis subyacente es que las redes neuronales, debido a su capacidad para identificar patrones complejos y modelar relaciones no lineales, pueden ser herramientas excepcionales para abordar este desafío.

El primer paso en nuestra metodología involucra la creación de una base de datos robusta. Esta base de datos incorpora observaciones experimentales junto con los coeficientes correspondientes del sistema de ecuaciones. La calidad y profundidad de estos datos son cruciales, ya que constituyen la base sobre la cual las redes neuronales aprenden y extraen insights significativos. La red neuronal es entrenada para reconocer patrones y correlaciones entre las variables experimentales y los coeficientes, lo que es esencial para predecir los parámetros del sistema de ecuaciones con precisión.

Para profundizar en nuestra comprensión, examinamos dos conjuntos de datos experimentales. Uno se enfoca en las trayectorias de las concentraciones de componentes dentro del Toggle Switch, mientras que el otro detalla el campo vectorial que las define. Este enfoque dual nos permite evaluar la robustez de nuestra metodología bajo diferentes perspectivas y conjuntos de datos. Además, se investigó cómo varía la eficacia de nuestro enfoque con diferentes niveles de resolución de los datos. Reducir gradualmente el tamaño del conjunto de datos nos permitió explorar los límites y la escalabilidad de nuestra metodología.

En términos de arquitectura de red neuronal, se exploraron varias estructuras. Una red densa fue nuestro punto de partida, seguida de una red con una función de costo personalizada, en la que la métrica de evaluación es derivada directamente de los datos experimentales. También se incluyó en el estudio una red neuronal convolucional, conocida por su eficacia en el procesamiento de patrones en datos estructurados espacialmente. La comparación entre estas arquitecturas nos brinda una perspectiva valiosa sobre su rendimiento relativo y adecuación para el problema en cuestión.

Un aspecto innovador y crucial de nuestra investigación fue la introducción de ruido en los datos. Esta estrategia no solo busca evaluar la robustez y fiabilidad de nuestras redes neuronales bajo condiciones realistas y menos ideales, sino que también resalta su capacidad superior en comparación con otras metodologías convencionales. Mientras que los enfoques tradicionales a menudo luchan o se ven significativamente afectados en presencia de datos ruidosos, las redes neuronales tienen una notable capacidad para manejar y aprender de estos datos imperfectos. El

manejo efectivo del ruido es un desafío notable en el modelado de sistemas biológicos, donde los datos experimentales pueden estar sujetos a una amplia gama de variabilidades y errores. Por lo tanto, la habilidad de las redes neuronales para extraer patrones útiles de conjuntos de datos ruidosos no solo es una ventaja técnica, sino que también representa un avance significativo en la modelación más realista y práctica de sistemas complejos.

La experimentación con datos ruidosos nos proporciona una comprensión más profunda de la estabilidad y la eficacia de nuestras predicciones, ofreciendo insights valiosos sobre la aplicabilidad de nuestra metodología en entornos experimentales donde las condiciones ideales raramente se cumplen.

Para evaluar la eficacia de los modelos, se utilizaron métricas especializadas y se realizó un análisis estadístico para cada conjunto de datos. Esta evaluación meticulosa es fundamental para comprender la precisión y la viabilidad de nuestras redes neuronales en la predicción de parámetros.

Finalmente, se desarrolló un modelo de red neuronal flexible y adaptable, capaz de procesar entradas de diferentes tamaños y, a pesar de estas variaciones, predecir con precisión los coeficientes del sistema de Toggle Switch. Este enfoque hacia un modelo 'único' ilustra la versatilidad y potencial de las redes neuronales en aplicaciones de modelado complejas.

Las secciones siguientes se dedicarán a detallar cada uno de estos componentes metodológicos, proporcionando así una comprensión integral del enfoque y las técnicas utilizadas en esta investigación innovadora.

2.2. Creación de base de datos deterministas

Con el objetivo de obtener una base de datos sólida y representativa, se parte del modelo original de Toggle Switch presentado en la ecuación [1.4]:

$$\begin{aligned} \frac{dX}{dt} &= \frac{a_1}{1+Y^n} - d_1 X + b_1 \\ \frac{dY}{dt} &= \frac{a_2}{1+X^n} - d_2 Y + b_2 \end{aligned} \tag{2.1}$$

Este modelo sirve como la base para generar un conjunto extenso de 10,000 instancias variando aleatoriamente los parámetros a_1, a_2, b_1, b_2, d_1, d_2, y n . Los parámetros a_1 y a_2 fueron tomados de 1 a 10, mientras que los parámetros d_1, d_2, b_1 y b_2 fueron tomados de 0 a 10. El parámetro n fue tomado de 1 a 5.

En el estudio del modelo de *Toggle Switch*, la elección de los rangos para los parámetros de simulación, específicamente tomando valores del 0 al 10 para la mayoría de los parámetros y del 1 al 5 para el parámetro n, puede justificarse desde una perspectiva biológica y matemática:

1. **Rango Biológico Realista:** En modelos de regulación genética como el *Toggle Switch*, es crucial que los parámetros reflejen rangos biológicamente realistas. Los parámetros como a_1, a_2, b_1, b_2, d_1, y d_2 (que representan tasas de producción y degradación, y constantes de activación/inhibición) suelen tener valores que reflejan la cinética de reacciones biológicas y la dinámica celular. Estos valores suelen estar dentro de un rango que refleja las concentraciones y tasas típicas observadas en células y tejidos. El rango de 0 a 10 puede representar, por ejemplo, concentraciones de moléculas (en unidades arbitrarias) o tasas de reacción normalizadas.

2. **Naturaleza del Parámetro n:** El parámetro n en el modelo *Toggle Switch* generalmente representa el coeficiente de Hill, que describe la cooperatividad de la unión de un regulador a su objetivo. En términos biológicos, refleja cómo la unión de una molécula afecta la probabilidad de unión de las siguientes moléculas. Los valores típicos de n suelen ser enteros pequeños (por ejemplo, 1, 2, 3, etc.), ya que representan el número de moléculas que se unen cooperativamente. Un valor de n mayor que 1 indica cooperatividad positiva, donde la unión de una molécula aumenta la probabilidad de unión de las siguientes. Un rango de 1 a 5 cubre la mayoría de los escenarios biológicos plausibles sin llegar a situaciones extremadamente no lineales o poco realistas.

3. **Consideraciones Matemáticas:** Desde una perspectiva matemática, elegir rangos apropiados para los parámetros asegura que el modelo se mantenga manejable y sus soluciones sean significativas. Un rango demasiado amplio podría conducir a comportamientos dinámicos que no son relevantes para el fenómeno que se está modelando o que resulten en un comportamiento numérico inestable durante la simulación.

4. **Comparabilidad y Consistencia:** Al establecer rangos específicos para los parámetros, se facilita la comparación entre diferentes simulaciones y experimentos. Esto ayuda a mantener la consistencia en el análisis y permite una interpretación más sencilla de los resultados.

Además, es importante destacar la razón por la que se eligieron los rangos de los parámetros a_1, a_2 y n para comenzar desde 1 en lugar de 0. Esta decisión se basa en que, si estos parámetros fueran iguales a 0, el sistema de ecuaciones se desacoplaría, resultando en dos ecuaciones diferenciales ordinarias independientes. Tal desacoplamiento implicaría una desviación del modelo de Toggle Switch, el cual se caracteriza precisamente por la interacción entre estas ecuaciones.

Este enfoque asegura una amplia cobertura del espacio de parámetros, permitiendo así un análisis exhaustivo de las dinámicas posibles dentro del marco del modelo de Toggle Switch. Utilizando estas instancias del modelo, se procedió a crear dos bases de datos sintéticas:

2.2.1. Base de datos de soluciones numéricas del sistema de ecuaciones

Esta base de datos almacena la información de las soluciones para las variables X e Y, que representan las concentraciones de las proteínas a lo largo del tiempo bajo diferentes condiciones iniciales.

En la construcción de la base de datos para nuestro estudio, se hizo uso de la función `solve_ivp`, proporcionada por la biblioteca SciPy de Python, específicamente dentro del módulo `scipy.integrate`. Esta función es reconocida por su capacidad para resolver sistemas de ecuaciones diferenciales ordinarias (ODEs) de manera eficiente y precisa. La elección de `solve_ivp` como nuestra herramienta de integración numérica fue motivada por su robustez y flexibilidad en el manejo de diversos tipos de ODEs.

Esta función permite la selección entre múltiples algoritmos de integración numérica. Para nuestro propósito, se utilizó el método 'RK5', que es adecuado para la mayoría de los problemas no rígidos y se basa en el conocido algoritmo de Runge-Kutta-Fehlberg.

La función `solve_ivp` fue utilizada para integrar cada instancia del sistema de ODEs desde un conjunto de condiciones iniciales variadas hasta un tiempo final predefinido, generando así trayectorias que reflejan el comportamiento dinámico de las variables de estado a lo largo del tiempo. Se prestó especial atención al vector de tiempo, el cual fue definido para garantizar que las soluciones se obtuvieran en intervalos temporales consistentes y uniformes a lo largo de todas las simulaciones.

Esta consistencia es crucial para el análisis posterior y la comparación entre diferentes trayectorias.

La implementación eficiente y el control preciso sobre los puntos de evaluación temporal brindados por `solve_ivp` resultaron en la generación de una base de datos extensa y detallada. Cada solución numérica generada enriqueció la base de datos, proporcionando una representación fidedigna del espacio de soluciones posible del modelo bajo estudio.

Se simuló cada sistema con 200, 100 y 50 puntos temporales y 10 conjuntos de condiciones iniciales diferentes, generando estructuras de datos de dimensiones $(10000, 10, 2, 200)$, $(10000, 10, 2, 100)$ y $(10000, 10, 2, 50)$, respectivamente (Figura 2.3). La estructura de los datos tiene la forma (m, l, k, j), donde:

- **m:** Cantidad de sistemas de ecuaciones diferenciales simulados
- **l:** Cantidad de condiciones iniciales para cada sistema
- **k:** Número de proteínas involucradas
- **j:**Número de puntos temporales en los que se resuelven cada sistema

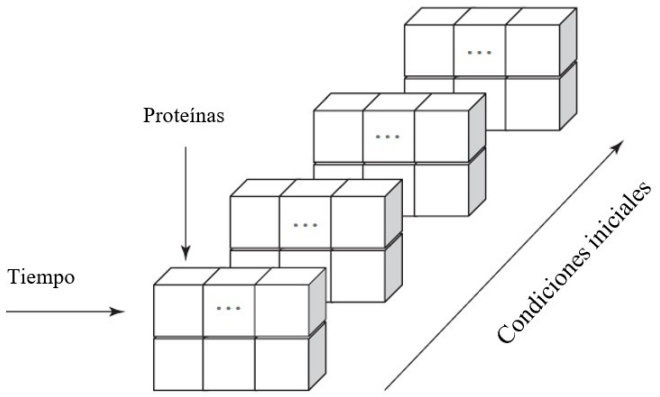


Figura 2.1: Estructura de cada una de las entradas de la base de datos que forman las soluciones.

Se eligió un intervalo temporal de 0 a 5 en las simulaciones del modelo y se fundamenta en observaciones empíricas y consideraciones prácticas. Específicamente, se ha determinado que, en la mayoría de los casos, las simulaciones alcanzan un estado de equilibrio o un comportamiento dinámico estable antes de sobrepasar el tiempo 5. Esta decisión se basa en varios factores clave:

1. **Observaciones de Equilibrio:** Mediante pruebas exhaustivas y análisis de simulaciones, se ha notado que los sistemas modelados tienden a estabilizarse o alcanzar un estado de equilibrio en un periodo menor a 5 unidades de tiempo. Esto sugiere que este intervalo es suficiente para capturar la dinámica esencial del sistema sin incurrir en cálculos redundantes o innecesarios.

2. **Consistencia en Análisis y Comparaciones:** Usar un intervalo temporal estandarizado de 0 a 5 en todas las simulaciones facilita la comparación directa entre diferentes conjuntos de parámetros y condiciones iniciales. Proporciona una base común para evaluar y contrastar los resultados obtenidos.

3. **Validación del Modelo:** Este rango temporal ha sido validado a través de múltiples simulaciones, demostrando su adecuación para capturar la dinámica del sistema bajo una variedad de condiciones. La consistencia en alcanzar el equilibrio dentro de este marco temporal refuerza la confianza en la validez del modelo.

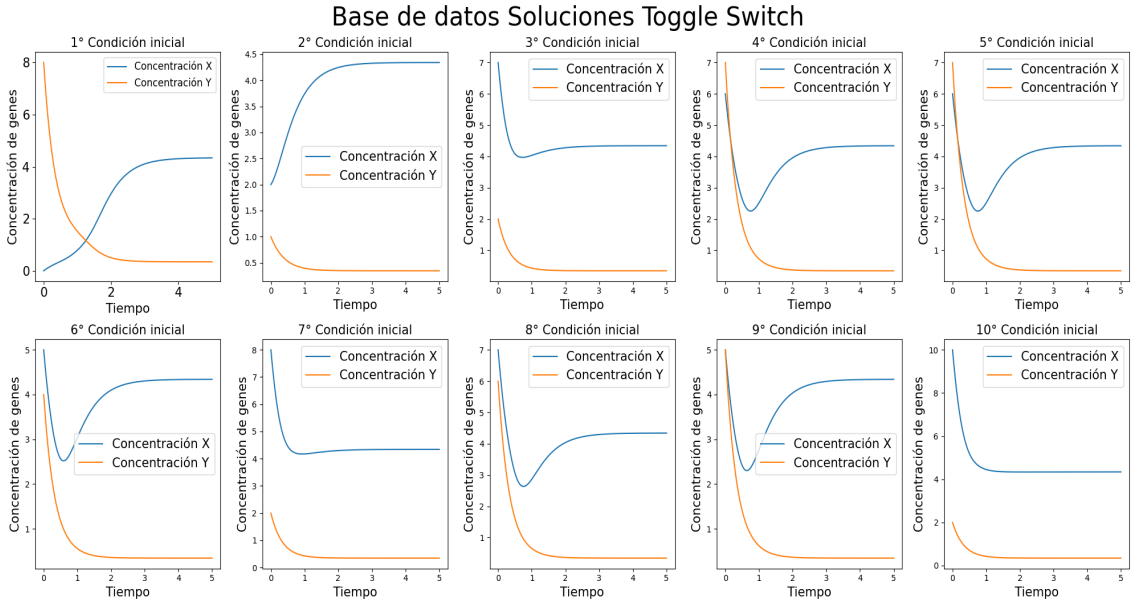


Figura 2.2: Un ejemplo tomado de la base de datos de las soluciones, se observan las soluciones del sistema de ecuaciones [2.1] con los siguientes parámetros: $a_1 = 8$, $a_2 = 3$, $b_1 = 1$, $b_2 = 1$, $d_1 = 2$, $d_2 = 3$, y $n = 3$ a diferentes condiciones iniciales.

2.2.2. Base de datos del campo vectorial asocial al sistema de ecuaciones

La segunda base de datos es un recurso clave en nuestra investigación, ya que encapsula información esencial sobre la dinámica del sistema de Toggle Switch. Esta base de datos se construyó a partir de regiones cuidadosamente seleccionadas del campo vectorial, las cuales emergen como soluciones al sistema de ecuaciones diferenciales. Esta selección permite una mirada profunda a cómo el sistema se comporta bajo diferentes condiciones y parámetros.

Para cada conjunto de parámetros dentro del modelo, se diseñó una cuadrícula de 10×10 en el espacio de fase. En esta cuadrícula, se muestrearon 100, 50 y 20 puntos en cada dimensión para las variables X e Y, representando distintos niveles de detalle y resolución. El resultado de este muestreo se traduce en bases de datos de dimensiones $(10000, 2, 100, 100)$, $(10000, 2, 50, 50)$ y $(10000, 2, 20, 20)$, cada una correspondiente a los diferentes niveles de resolución de muestreo mencionados anteriormente. Estas dimensiones representan un compendio de datos que varían desde una resolución muy alta hasta una más reducida, proporcionando así una plataforma rica para probar la eficacia y sensibilidad de las redes neuronales en diferentes contextos de datos.

En términos de su estructura, cada base de datos sigue el formato (m, l, k, j), donde:

- **m:** Cantidad de sistemas de ecuaciones diferenciales

- **l:** En la profundidad se guardan los valores del espacio fase en la dirección x y y, es decir $l = 2$

- **k:** Total de puntos en los que se evalúa en la dirección x
- **j:** Total de puntos en los que se evalúa en la dirección y

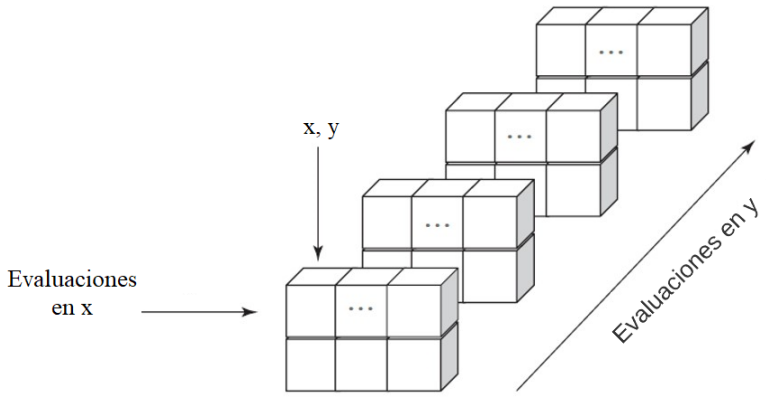


Figura 2.3: Estructura de cada una de las entradas de la base de datos que forman los campos vectoriales.

Un campo vectorial es una representación gráfica que asigna un vector a cada punto en el espacio de fases. En sistemas dinámicos, el campo vectorial describe la velocidad y dirección del sistema en cada estado posible [89]. Para obtener un campo vectorial a partir de un sistema de ecuaciones diferenciales, se sigue el procedimiento detallado a continuación:

1. **Definir el Sistema de Ecuaciones Diferenciales:** Considere un sistema de dos dimensiones con las siguientes ecuaciones:

$$\frac{dX}{dt} = f(X, Y), \tag{2.2}$$

$$\frac{dY}{dt} = g(X, Y), \tag{2.3}$$

   donde f y g son funciones que determinan las tasas de cambio de las variables de estado X e Y respectivamente.

2. **Evaluar las Funciones en el Espacio de Fases:** Para cada punto en el espacio de fases, se calculan las funciones f y g, que proporcionan las componentes del vector en ese punto.

3. **Construir los Vectores:** Las derivadas obtenidas, $(\frac{dX}{dt}, \frac{dY}{dt})$, se transforman en vectores que indican la dirección y magnitud del cambio en el sistema en cada punto del espacio de fases [90].

4. **Representar Gráficamente:** Estos vectores se visualizan como flechas en un diagrama, cada una con su origen en el punto correspondiente y con longitud y dirección que representan la magnitud y dirección del vector. La colección completa de estas flechas conforma el campo vectorial.

La representación del campo vectorial proporciona una comprensión visual e intuitiva del flujo del sistema a través del tiempo y es especialmente útil en la modelización de fenómenos biológicos para ilustrar cómo las concentraciones de distintas sustancias evolucionan a lo largo del tiempo bajo diversas condiciones.

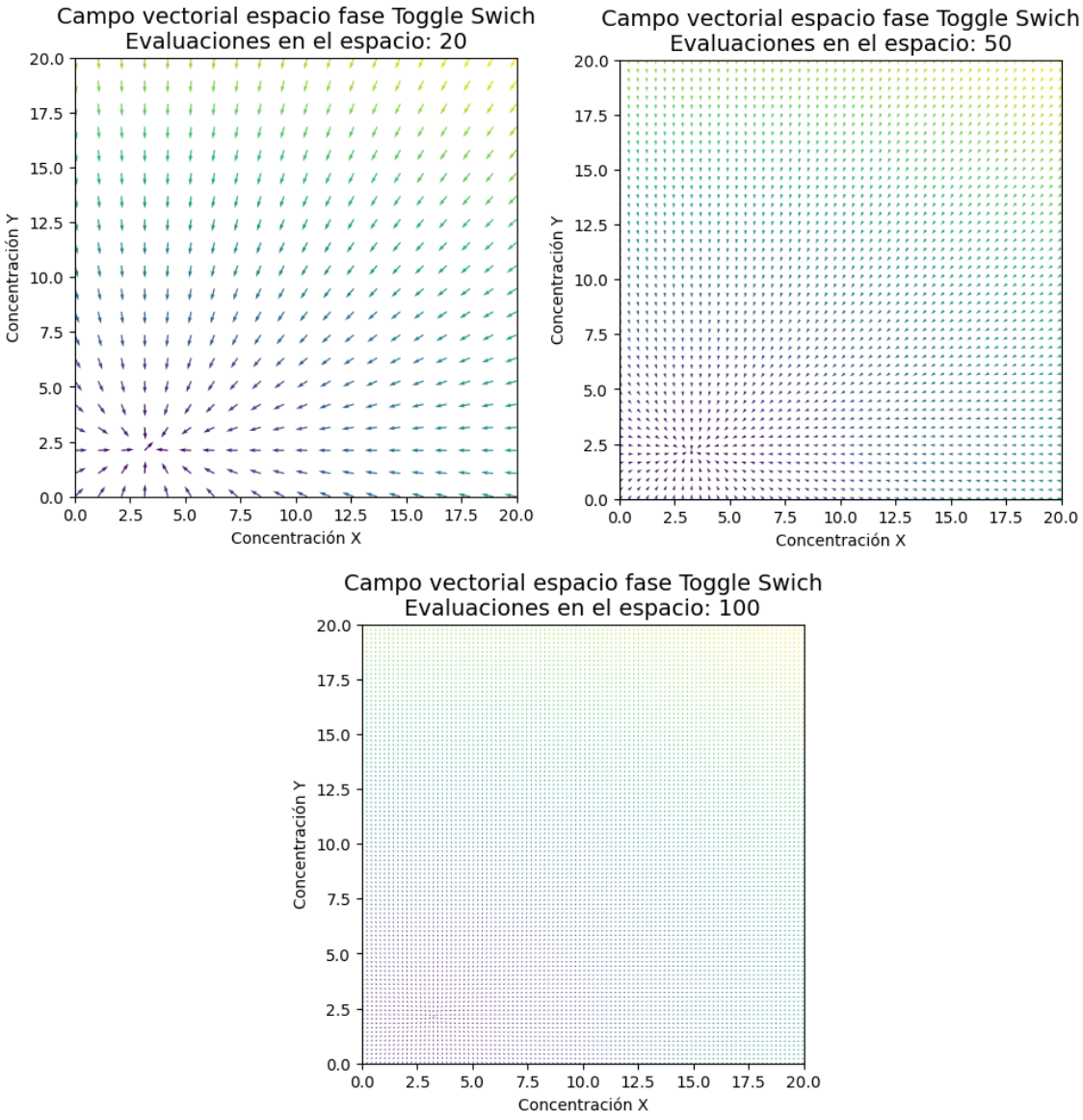


Figura 2.4: Un ejemplo tomado de la base de datos de los campos vectoriales, se observan el campo vectorial sistema de ecuaciones [2.1] con los siguientes parámetros: $a_1 = 10$, $a_2 = 2$, $b_1 = 4$, $b_2 = 7$, $d_1 = 3$, $d_2 = 4$, y $n = 3$ a diferentes resoluciones. De izquierda a derecha 100, 50 y 20 evaluaciones por eje.

La decisión de establecer el rango del campo vectorial de 0 a 10 se basa en una serie de consideraciones prácticas y observacionales. Este rango se ha elegido cuidadosamente para asegurar que se capturen adecuadamente las características dinámicas del sistema, al tiempo que se mantiene la relevancia biológica y la eficiencia computacional.

1. **Consistencia y Comparabilidad:** Al igual que con la elección del intervalo temporal, el uso de un rango estandarizado para el campo vectorial permite una comparación coherente y directa entre diferentes simulaciones. Esto facilita la interpretación de los resultados y su aplicación en diversos escenarios experimentales.

2. **Observaciones Empíricas:** Las pruebas preliminares han mostrado que este rango es suficiente para observar los patrones de comportamiento críticos del modelo, incluyendo la estabilidad, las transiciones y las bifurcaciones, proporcionando así una comprensión integral del sistema.

3. **Validación del Modelo:** Este rango ha sido validado mediante una serie de simulaciones que han demostrado su capacidad para captar fielmente la dinámica del sistema, reforzando la precisión y fiabilidad del modelo.

En ambos casos (soluciones numéricas y campo vectorial), se guardan los valores de a_1, a_2, d_1, d_2, b_1, b_2 y n que se utilizaron para generar los sistemas de ecuaciones diferenciales. Estos parámetros son cruciales, ya que no solo dictan la dinámica del sistema modelado, sino que también constituyen la piedra angular para la fase posterior de nuestra investigación: la evaluación y el ajuste de la precisión de las redes neuronales. La retención de estos valores es imperativa para el entrenamiento supervisado de las redes, donde funcionan como la verdad fundamental contra la cual se compara la salida de la red, permitiendo así la optimización de la arquitectura de la red y sus hiperparámetros en base a datos fidedignos.

Para la implementación y evaluación de la red neuronal, las bases de datos generadas fueron divididas en tres conjuntos distintos, con el fin de entrenar el modelo y evaluar su capacidad predictiva y de generalización:

- **Conjunto de Entrenamiento (7000 muestras)**: Este conjunto es la columna vertebral del proceso de aprendizaje de la red. Las 7000 muestras se utilizan para ajustar los parámetros y los pesos de la red, permitiendo que el modelo aprenda directamente de los datos.

- **Conjunto de Validación (2000 muestras)**: Este conjunto actúa como un control durante el entrenamiento. Se utiliza para monitorear el aprendizaje y ajustar la arquitectura de la red y los hiperparámetros. Su objetivo es prevenir el sobreajuste, asegurando que la red no memorice los datos de entrenamiento, sino que aprenda patrones generalizables.

- **Conjunto de Prueba (1000 muestras)**: Constituye la prueba definitiva de la eficacia del modelo. Estas muestras no se utilizan durante el entrenamiento, por lo que ofrecen una evaluación imparcial del rendimiento del modelo en datos completamente nuevos. Este conjunto es indicativo de cómo se espera que la red neuronal se comporte en aplicaciones reales y situaciones no vistas.

Esta estrategia de partición de los datos asegura que el modelo pueda ser entrenado de forma efectiva y evaluado de manera imparcial, proporcionando así una medida confiable de su rendimiento. Como resultado, se conformaron un total de 18 bases de datos junto con sus respectivos coeficientes, lo que constituye un recurso integral para el entrenamiento, la validación y las pruebas de la red neuronal desarrollada.

Todos los conjuntos de datos mencionados anteriormente, junto con sus correspondientes coeficientes, han sido almacenados en archivos con formato `.npy`, un tipo de archivo binario para almacenar arrays de Numpy con extensión propia de la librería Numpy de Python. Este formato es particularmente eficiente para la escritura y lectura de datos, permitiendo un manejo rápido y un acceso directo a los arrays de grandes dimensiones, lo cual es esencial para el manejo eficiente de los datos durante el entrenamiento y la evaluación de la red neuronal.

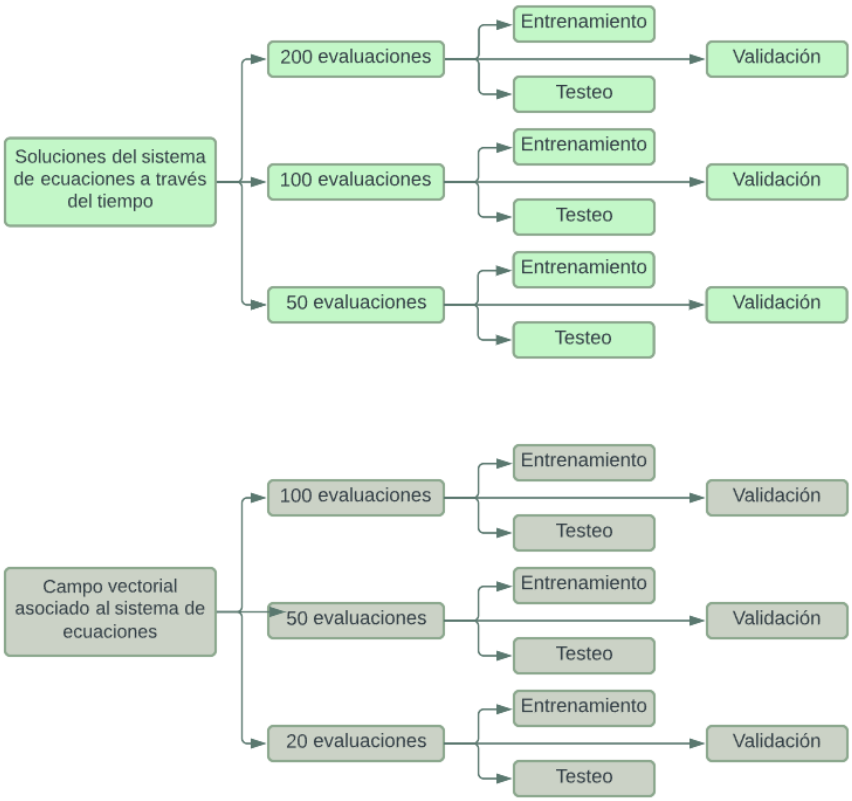


Figura 2.5: Esquema que muestra cómo están organizadas cada una de las bases de datos

2.2.3. ¿Por qué se seleccionaron las bases de datos de esta manera?

Se eligió el tamaño de la base de datos de manera que fuera lo suficientemente grande para representar una muestra significativa de los datos, permitiendo que las redes neuronales se entrenen de la mejor y más generalizada manera posible. Al mismo tiempo, se tuvo en cuenta que, a mayor tamaño de los datos, mayor sería el costo computacional al utilizarlos en una red neuronal. A continuación, se presenta un ejemplo del tamaño de los datos:

La base de datos, que consta de valores de entrenamiento para las trayectorias de las soluciones del sistema de ecuaciones diferenciales evaluadas en 200 puntos en el tiempo, tiene una dimensión de (7000, 10, 2, 200), siendo estos objetos float. Un objeto float en Python ocupa 8 bytes en la memoria, lo que en este caso equivale a $7000 \times 10 \times 2 \times 200 = 28,000,000$ *valores float. Al multiplicar estos valores por los 8 bytes que ocupan de espacio, obtenemos 224,000,000 bytes. Para convertir bytes a megabytes (MB), se debe dividir la cantidad de bytes entre* (1024×1024)*, ya que hay 1024 bytes en un kilobyte (KB) y 1024 KB en un MB. Por lo que:*

$$\frac{224,000,000\ bytes}{1024 \times 1024} \approx 213{,}623\ MB$$

Si en lugar de 10,000 sistemas de ecuaciones se hubieran elegido 100,000, el tamaño de la base de datos habría sido 10 veces mayor, alcanzando un tamaño de 2.086 GB (1 GB = 1024 MB). Este tamaño resulta demasiado grande para un procesamiento eficiente de redes neuronales.

2.3. Creación de redes neuronales datos deterministas

Para la implementación y diseño de las redes neuronales en este estudio, se recurrió al uso de las bibliotecas de código abierto TensorFlow y Keras, que son ampliamente reconocidas en la comunidad científica por su eficacia y flexibilidad en el desarrollo de modelos de aprendizaje profundo. Se concibieron arquitecturas de redes neuronales específicas para cada una de las bases de datos generadas, con el propósito de adaptar la estructura de la red a la naturaleza de los datos y al problema específico a resolver.

La estrategia para el diseño de las redes se diversificó aún más al crear tres variantes de arquitecturas para cada conjunto de datos:

- **Red Neuronal Densamente Conectada (DNN):** Una arquitectura clásica compuesta por múltiples capas densas que facilitan la captura de relaciones no lineales complejas entre las variables de entrada y salida.
- **Red Neuronal Densamente Conectada con Función de Costo Personalizada:** Una variante de la red DNN donde se incorpora una función de costo diseñada específicamente para mejorar el aprendizaje con respecto a ciertas características del sistema modelado.
- **Red Neuronal Convolucional (CNN):** Se explora el uso de arquitecturas convolucionales, particularmente adecuadas para el procesamiento de datos estructurados en forma de rejillas, como imágenes o, en nuestro caso, matrices que representan campos vectoriales o secuencias temporales de datos.

Independientemente del tipo de red seleccionada, el proceso de desarrollo comienza de manera uniforme, siguiendo un conjunto de pasos estandarizados:

1. **Importamos librerías** En el marco del desarrollo de código para este proyecto, se inicia con la importación de librerías fundamentales que facilitarán la construcción de la red neuronal y el manejo de datos de forma óptima. 'TensorFlow', alias 'tf', y 'Keras' son centrales para el modelado y entrenamiento de algoritmos de aprendizaje profundo. 'pathlib' se emplea para la manipulación eficiente de las rutas de archivos, mientras que 'os' permite la interacción con el sistema de archivos. La visualización de datos se logra a través de 'matplotlib.pyplot', abreviado como 'plt'. 'pandas' y 'numpy', importados como 'pd' y 'np' respectivamente, son esenciales para el análisis de datos y operaciones matemáticas avanzadas.

   En cuanto a la arquitectura de la red neuronal, se importa 'Sequential' de 'tensorflow.keras' para la construcción de modelos en forma de secuencia de capas. Las capas de 'Conv2D', 'MaxPooling2D', 'AveragePooling2D', 'Dense', 'Flatten', 'Dropout' y 'BatchNormalization' de 'tensorflow.keras.layers' se seleccionan para estructurar la red, ofreciendo una variedad de operaciones, desde la convolución hasta la normalización por lotes. Además, 'HeNormal' de 'tensorflow.keras.initializers' se utiliza para la inicialización ponderada, y 'l1' y 'l2' de 'keras.regularizers' se integran para aplicar regularización, crucial para evitar el sobreajuste en el aprendizaje de la red.

2. **Importamos los datos.** En esta parte del código, procedemos a cargar las bases de datos requeridas para entrenar, probar y validar cada red neuronal. Por ejemplo, en el caso de los campos vectoriales que se evalúan en 100 puntos distintos, se importan seis conjuntos de datos diferentes. Esto incluye un conjunto de datos de entrenamiento con dimensiones (7000, 2, 100, 100) acompañados de sus respectivos coeficientes de dimensión (7000, 7). De forma similar, se cargan los datos de prueba y sus coeficientes con dimensiones de (2000, 2, 100, 100) y (2000, 2), respectivamente. Por último, se importan los datos de validación y sus coeficientes, los cuales poseen dimensiones de (1000, 2, 100, 100) y (1000, 2). Esta

estructuración detallada de las bases de datos es fundamental para un análisis exhaustivo y una validación efectiva de las redes neuronales en estudio.

Para optimizar el funcionamiento de las redes convolucionales, se realiza una reordenación de las dimensiones en los conjuntos de datos. Este paso crítico consiste en transponer los datos de manera que las dimensiones de mayor magnitud se sitúen como primer argumento, lo que incrementa la flexibilidad durante el proceso de convolución. Tomemos como ejemplo el conjunto de datos de entrenamiento para los campos vectoriales evaluados en 100 puntos, que originalmente poseen la forma (7000, 2, 100, 100). Estos se reorganizan para adoptar la configuración (7000, 100, 100, 2). Análogamente, para la base de datos correspondiente a las trayectorias evaluadas en 200 puntos temporales, cuya forma inicial es (7000, 10, 2, 200), se ajustan para que su nueva forma sea (7000, 200, 10, 2).

3. **Unión de los coeficientes con los datos.** Para la preparación de nuestros conjuntos de datos de entrenamiento, prueba y validación, optamos por la función `from_tensor_slices` provista por TensorFlow. Esta función es fundamental para la creación de un dataset que particiona los tensores en segmentos a lo largo de su primera dimensión, lo que facilita la correspondencia exacta entre datos y etiquetas.

   La estructura de los datos resultantes para el caso de los campos vectoriales evaluados en 100 puntos es la siguiente:

```
    <TensorSliceDataset element_spec=(
    TensorSpec(shape=(2, 100, 100), dtype=tf.float64, name=None),
    TensorSpec(shape=(7,), dtype=tf.int32, name=None)
)>
```

4. **Mezclar y procesar por lotes los conjuntos de datos.**Los conjuntos de datos se barajan y se organizan en batches (lotes) para su procesamiento. Hemos seleccionado un tamaño de lote de 100 unidades tras considerar un equilibrio entre la eficiencia computacional y la capacidad de generalización del modelo.

2.3.1. Optimización de hiperparámetros con Optuna

Al comienzo de este estudio, las redes neuronales se establecieron utilizando un enfoque de selección de hiperparámetros basado en conocimientos previos y experimentación heurística, lo que nos permitió obtener un modelo inicial para abordar el complejo problema inverso de la regulación genética. Esta aproximación inicial ayudó a establecer una comprensión preliminar sobre cómo la estructura de los datos y la configuración de la red podrían influir en el proceso de entrenamiento. No obstante, la complejidad y la alta dimensionalidad del problema nos llevaron a buscar un método más eficiente y sistemático para la optimización de hiperparámetros.

La integración de Optuna, un optimizador automático de hiperparámetros, se convirtió en una parte crucial de la refinación del modelo. Con Optuna, pudimos explorar de manera más exhaustiva el espacio de hiperparámetros y ajustar nuestras redes neuronales para mejorar la resolución del problema inverso. Optuna se destaca por su capacidad para automatizar la búsqueda y por su eficiencia al evaluar una amplia gama de configuraciones posibles, una tarea que sería inviable realizar manualmente. La optimización con Optuna ha perfeccionado la estructura de nuestros modelos predictivos, mejorando sustancialmente la obtención de los coeficientes que definen a nuestros sistemas dinámicos.

Se explicará brevemente la estructura de las redes sin Optuna y de manera detallada la estructura de las redes con Optuna,incluyendo el código que dio origen a estas estructuras mejoradas.

2.3.2. Métricas

En nuestro modelo, se ha diseñado una métrica personalizada para evaluar con precisión la capacidad predictiva del modelo con respecto a los parámetros específicos del sistema de ecuaciones diferenciales. Esta métrica se basa en calcular el error medio absoluto (MAE) de manera individual para cada parámetro y luego promediar estos errores para obtener una visión global de la precisión del modelo.

El MAE es una medida estadística que se utiliza para cuantificar la diferencia entre valores observados y predicciones. Se define como el promedio de las diferencias absolutas entre las predicciones y los valores verdaderos, matemáticamente expresado como:

$$MAE_i = \frac{1}{N} \sum_{j=1}^{N} |y_{ji} - \hat{y}_{ji}| \tag{2.4}$$

donde i representa el índice correspondiente a cada parámetro del sistema, N el número total de muestras, y_{ji} el valor verdadero y $\hat{y}_{ji}$ el valor predicho para el parámetro i en la muestra j.

Para obtener una medida comprensiva del rendimiento general del modelo, desarrollamos la métrica 'mean_metric', que promedia los MAE de todos los parámetros, ajustados por la suma de los valores verdaderos de los parámetros en cada muestra, proporcionando así una evaluación normalizada:

$$\text{mean\_metric} = \frac{\sum_{i=1}^{M} MAE_i}{\sum_{i=1}^{M} |y_i|} \tag{2.5}$$

donde M es la cantidad total de parámetros del sistema.

La implementación de estas métricas permite un análisis detallado del rendimiento del modelo en cada parámetro predicho, otorgando así un entendimiento más profundo de la precisión del modelo en la estimación de los parámetros que rigen el comportamiento del sistema modelado.

La eficiencia de la red neuronal se mide principalmente a través de la métrica `mean_metric`, que proporciona una valoración promedio del error en todos los coeficientes, ajustado por la suma de los coeficientes verdaderos. Este enfoque nos permite identificar qué tan bien está realizando predicciones el modelo respecto a los valores reales de los coeficientes

Además, al final de cada época de entrenamiento, se puede visualizar que tan bien o mal se están prediciendo cada uno de los coeficientes. Esto es crucial para comprender el comportamiento del modelo y determinar qué coeficientes están siendo aprendidos con mayor precisión y cuáles presentan desafíos. La siguiente salida de una época de entrenamiento ilustra cómo se van comportando cada uno de los componentes en el proceso:

```
Epoch 527/1000
70/70 [==============================] - 2s 23ms/step - loss: 0.5834 - a_1: 0.4046
    - a_2: 0.3986 - b_1: 0.3614 - b_2: 0.3777 - d_1: 0.3472 - d_2: 0.3536 - n:
    0.3466 - mean_metric: 0.0808 - val_loss: 0.3861 - val_a_1: 0.1583 - val_a_2:
    0.1612 - val_b_1: 0.1639 - val_b_2: 0.1688 - val_d_1: 0.1408 - val_d_2: 0.1384
    - val_n: 0.1269 - val_mean_metric: 0.0330 - lr: 5.3140e-04
```

La salida refleja cómo el modelo está aprendiendo y ajustándose a los datos. Al observar los valores de las métricas por coeficiente, podemos identificar patrones y realizar ajustes para mejorar el entrenamiento del modelo.

2.3.3. Optimizador

Se determinó excluir al optimizador de la búsqueda de hiperparámetros realizada por Optuna debido a que, en las evaluaciones preliminares, Adam constantemente emergía como el optimizador más eficaz. Por consiguiente, este fue seleccionado por defecto para el entrenamiento de nuestras redes neuronales.

Adicionalmente, se implementó un valor de recorte (clipvalue) de 100 en el optimizador Adam para prevenir el fenómeno conocido como 'gradiente explosivo'. Estos suceden cuando los gradientes se vuelven muy grandes, lo que puede hacer que la optimización sea inestable y los parámetros de la red neuronal cambien de manera drástica. Este enfoque consiste en limitar el valor absoluto de los gradientes a un máximo especificado, garantizando así que durante el entrenamiento, cualquier gradiente que supere este umbral sea adecuadamente escalado para evitar oscilaciones excesivas o divergencias en los pesos de la red [58].

2.3.4. Tasa de aprendizaje adaptativa

Optamos por no incluir la tasa de aprendizaje en el ámbito de optimización de Optuna. En su lugar, se prefirió adoptar una estrategia de tasa de aprendizaje adaptativa que evoluciona a lo largo de las épocas de entrenamiento. Inicialmente, se estableció una tasa de aprendizaje de 0,001, la cual se redujo linealmente hasta alcanzar 0,00001 en la época 1,000. Este ajuste dinámico de la tasa de aprendizaje se diseñó para responder al comportamiento del modelo en términos de sobreajuste, comenzando la reducción en la época donde se observaran las primeras señales de este. De este modo, la tasa de aprendizaje se adaptaba en función del desempeño y la capacidad de generalización del modelo a lo largo del proceso de entrenamiento.

En el proceso de entrenamiento de los modelos de aprendizaje profundo, la tasa de aprendizaje es un hiperparámetro crítico que influye directamente en la velocidad y la eficacia con la que el modelo converge hacia un mínimo en la función de pérdida. Se implementó una estrategia de tasa de aprendizaje adaptativa, que ajusta dinámicamente este hiperparámetro a lo largo de las épocas de entrenamiento. La lógica detrás de esta estrategia se describe a continuación:

- Se inicia con una **tasa de aprendizaje inicial** (α_{inicial}) de 0.001. Este es el valor con el que comienza el entrenamiento.
- La **tasa de aprendizaje final** (α_{final}), el valor al cual la tasa de aprendizaje debe converger al final del entrenamiento, se establece en 0.00001.
- El proceso de disminución comienza en la **época de inicio de disminución** (ϵ_{inicio}), que varía de acuerdo al experimento, esto debido a que en algunos modelos se empieza a sobreajustar antes, por lo que se empieza a disminuir justo antes de que esto pase.
- La disminución continúa hasta la **época de fin de disminución** (ϵ_{fin}), que se establece en la época 1000.
- La función define tres comportamientos condicionales basados en la época actual (ϵ):
  1. Si $\epsilon < \epsilon_{\text{inicio}}$, se mantiene α_{inicial}.
  2. Si $\epsilon_{\text{inicio}} \leq \epsilon \leq \epsilon_{\text{fin}}$, se calcula y aplica una **tasa de aprendizaje adaptativa** que disminuye linealmente de α_{inicial} a α_{final} a lo largo del intervalo de disminución.
  3. Si $\epsilon > \epsilon_{\text{fin}}$, la tasa de aprendizaje se fija en α_{final}.

Este enfoque adaptativo permite afinar el modelo de manera más efectiva, proporcionando una disminución controlada de la tasa de aprendizaje que se alinea con la evolución del proceso de entrenamiento.

2.3.5. Función de costo

Dentro del marco de nuestro estudio, la función de costo desempeña un papel crucial al dirigir el proceso de aprendizaje de la red neuronal. La meta es predecir con precisión los coeficientes de los sistemas de ecuaciones diferenciales a partir de los datos proporcionados por los campos vectoriales y las trayectorias. Por esta razón, se seleccionó la función de costo *mean squared error* (MSE) para todas las variantes de redes neuronales implementadas.

En Keras, la función de pérdida MSE se define como el promedio de los cuadrados de las diferencias entre los valores predichos por la red neuronal y los valores verdaderos. Matemáticamente, para un conjunto de N predicciones, el MSE se calcula como:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.6}$$

donde y_i representa el valor verdadero y $\hat{y}_i$ es el valor predicho por la red neuronal para el i-ésimo coeficiente.

La elección de MSE como función de costo se justifica por su eficacia en la cuantificación del error cuadrático promedio entre los coeficientes predichos por la red y los valores reales. Este enfoque es particularmente beneficioso en contextos de regresión, como el nuestro, donde es imperativo minimizar la discrepancia entre las predicciones y los valores verdaderos. Al elevar al cuadrado las diferencias individuales, la función MSE penaliza de manera más significativa los errores grandes, lo que es deseable cuando se requiere que la red neuronal prediga los coeficientes con alta precisión.

El empleo de MSE está motivado por su tendencia a producir estimaciones más precisas y su capacidad para facilitar la convergencia del gradiente durante el entrenamiento. Al minimizar el error cuadrático medio, se impulsa a la red a afinar sus pesos y sesgos de tal forma que los coeficientes generados se alineen estrechamente con los datos subyacentes que definen el comportamiento dinámico de los sistemas modelados. Por lo tanto, MSE se alinea con nuestro objetivo de desarrollar un modelo que no solo pueda predecir con precisión sino que también refleje una comprensión profunda de las complejidades inherentes a los sistemas de ecuaciones diferenciales en estudio.

A partir de este punto, las metodologías adoptadas para cada tipo de red neuronal comienzan a diferenciarse, siguiendo trayectorias específicas y adaptadas a sus respectivas arquitecturas y funciones. Cada enfoque está diseñado para capitalizar las fortalezas inherentes a la configuración particular de la red, garantizando así que los procesos de entrenamiento y validación estén óptimamente alineados con los objetivos de aprendizaje y predicción establecidos.

2.4. Red neuronal densa

En el caso de las redes neuronales densas, se adoptaron diversas estructuras de red específicamente adaptadas a la naturaleza de los datos, diferenciando entre trayectorias y campos vectoriales, para la configuración inicial sin Optuna. Esta diferenciación se basó en la premisa de que las características únicas de cada tipo de datos requerían arquitecturas especializadas. Por otro lado, al introducir Optuna en nuestro proceso, se implementó un sistema de búsqueda de hiperparámetros meticulosamente diseñado para afinar y optimizar las redes, independientemente de la tipología de los datos. A continuación, se detallará el mecanismo de búsqueda de hiperparámetros utilizado para el caso con Optuna:

1. **Definición de la Función Objetivo:** Se define la función objective(trial) que guía el proceso de optimización, encargada de construir y evaluar la red neuronal basada en un conjunto de hiperparámetros propuestos.

2. **Construcción del Modelo:** Se inicia con una red neuronal secuencial a la que se le añade una capa Flatten para aplanar las dimensiones de entrada.

3. **Bucle de Capas Densas:** Se crea un número variable de capas densas, con el número de capas, número de unidades, función de activación, inicializador de kernel y regularizador de kernel sugeridos por Optuna. Estos hiperparámetros son seleccionados dentro de un rango predefinido y basados en el rendimiento del modelo.

4. **Regularización:** Se aplica regularización L1 o L2 en las capas densas según lo determinado por Optuna, con un valor de 0.001 para el parámetro de regularización.

5. **Normalización y Dropout:** Después de cada capa densa, se decide si añadir una capa de BatchNormalization y/o dropout, también con parámetros sugeridos por Optuna. La tasa de dropout varía entre 0.1 y 0.5.

6. **Capa de Salida:** Se añade una capa de salida con 7 unidades.

7. **Entrenamiento:** Se entrena el modelo usando los datos de entrenamiento y validación con un total de 50 épocas. Aunque no sean las 1000 épocas que se entrena la estructura ganadora, nos ayuda a poder ver cual modelo sigue el mejor camino y podemos evitar tiempos computacionales exageradamente grandes.

8. **Resultados de la Validación:** Se extrae métrica $mean\_metric$ del historial de entrenamiento para usarla como valor de retorno, que Optuna intentará minimizar.

9. **Optimización con Optuna:** Se crea un estudio con la dirección de minimización y se ejecutan 300 ensayos para encontrar la mejor configuración de hiperparámetros.

10. **Salida de Resultados:** Se imprimen los resultados del estudio, incluido el número de ensayos completados y los detalles del mejor ensayo con su valor y parámetros óptimos.

Este enfoque de búsqueda de hiperparámetros permite afinar de manera eficiente las redes neuronales densas, buscando el equilibrio óptimo entre capacidad predictiva y complejidad del modelo.

2.4.1. Trayectorias

Sin Optimización de Optuna

Para el caso de las trayectorias, se detalla a continuación las arquitecturas iniciales de las redes neuronales desplegadas en nuestros experimentos previos a la optimización con Optuna. Las redes fueron diseñadas utilizando el módulo `Sequential` de TensorFlow, que facilita la construcción de modelos mediante la adición secuencial de capas. Esta estructura lineal simplifica el proceso de configuración y ajuste del modelo, permitiendo una concatenación ordenada y clara de capas con distintas funciones y propósitos. La arquitectura de la red para el caso de 200 evaluaciones temporales, previa a la optimización hiperparamétrica, es la siguiente:

- Se compenza con una **capa de aplanamiento** (Flatten) que transforma los datos de entrada de un tensor tridimensional con forma (10, 2, 200) en un vector unidimensional, preparando los datos para el procesamiento en capas densas.

- Posteriormente se añade una capa de **normalización por lotes** (BatchNormalization) que se emplea para estabilizar y acelerar el entrenamiento al normalizar las activaciones de la capa anterior, aplicando la normalización por lotes en cada paso de entrenamiento.
- Se incorpora la **primera capa densa** compuesta por 200 neuronas, que utiliza la función de activación 'tanh'. Esta capa está equipada con una inicialización de pesos HeNormal, que favorece la convergencia en redes profundas, y se aplica una regularización L1 con un coeficiente de 0.001 para promover la simplicidad del modelo y reducir el sobreajuste.
- La **segunda capa densa** cuenta con 100 unidades y sigue utilizando la función de activación 'tanh', combinada con la misma inicialización de pesos HeNormal y regularización L1 de 0.001.
- La **tercera capa densa** se configura con 50 unidades, manteniendo la activación 'tanh' y la estrategia de inicialización y regularización de las capas anteriores. La disminución progresiva del número de neuronas ayuda a compactar la información y prepara la red para la consolidación de las características aprendidas.
- Finalmente, se añade la **cuarta capa densa** con 100 neuronas, cambiando la función de activación a 'relu'. Esta función introduce una dinámica diferente en la red, aportando robustez y mejorando la capacidad de la red para modelar relaciones no lineales. Se continúa con la inicialización HeNormal y la regularización L1 de 0.001, manteniendo así la coherencia en la regularización del modelo.
- Se añade **una capa de 'Dropout'** con una tasa del 20 % después de la 4^a capa densa, que proporciona una técnica de regularización adicional, descartando aleatoriamente un subconjunto de activaciones en la capa para promover la robustez del modelo.
- Se añade una última capa de **BatchNormalization** después del Dropout y antes de la capa de salida.
- Por último, **la capa de salida**, compuesta por 7 neuronas, cada una correspondiente a un coeficiente del sistema de ecuaciones diferenciales, para producir la salida final del modelo.

En la adaptación de la red neuronal para el análisis de datos con 100 evaluaciones temporales, se realizaron ajustes menores en la arquitectura para adaptarse a la disminución de la dimensión temporal. El cambio más notable fue la modificación del `input_shape` en la primera capa de aplanamiento (`Flatten`), que se ajustó a (10, 2, 100) para reflejar la nueva forma de los datos de entrada. Adicionalmente, se efectuó una reducción proporcional en el número de neuronas de las capas `Dense` para mantener una coherencia con la cantidad reducida de datos temporales. Las cantidades de neuronas en las capas densas se ajustaron a 100, 50, 25 y 50, respectivamente, lo cual representa una disminución a la mitad en comparación con la configuración inicial para 200 evaluaciones temporales.

Mientras que para el conjunto de datos correspondiente a 50 evaluaciones temporales, se procedió con una adaptación adicional de la arquitectura de la red neuronal para adecuarse eficientemente a la menor granularidad temporal. Esta modificación implicó una nueva configuración del `input_shape` en la capa `Flatten`, que se estableció en (10, 2, 50) para alinearse con la entrada disminuida. Asimismo, se llevó a cabo una reducción adicional en el tamaño de las capas `Dense`, disminuyendo el número de neuronas a 50, 25, 12 y 25 respectivamente para cada una de ellas.

Este ajuste en la densidad de las capas neuronales no solo se alinea con la reducción de la dimensión temporal de los datos sino que también mantiene la coherencia en términos de complejidad del modelo y capacidad de procesamiento. Al reducir el número de neuronas, se busca optimizar el uso de los recursos computacionales y evitar el riesgo de sobreajustar el modelo a un conjunto de datos más limitado en términos de volumen temporal, manteniendo así la eficacia y la generalización del aprendizaje.

Con Optimización de Optuna

Con Optuna, las redes siguieron una estructura completamente diferente, sin seiguir patrones predeterminados como fue el caso pasado. Para el caso de 200 evaluaciones temporales Optuna nos arrojó la siguiente estructura de red:

- Se inicia con la **creación del modelo**, instanciando la clase `Sequential` de Keras.
- Se introduce una **capa de aplanamiento** (`Flatten`) para transformar la entrada en un formato adecuado para procesamiento denso, específicamente de una forma de $10 \times 2 \times 200$.
- Se configura la primera **capa densa** con 51 unidades, activación ReLU, inicializador de pesos `HeNormal` y regularización L2.
- La segunda **capa densa** se define con 74 unidades, activación ReLU, inicializador de pesos `random_normal` y regularización L1.
- A continuación, se añade una **capa de normalización por lotes** (`BatchNormalization`) después de la segunda capa densa para estabilizar y acelerar el entrenamiento.
- La tercera **capa densa** incluye 102 unidades, activación sigmoide, inicializador de pesos `random_normal` y regularización L2.
- Después de la tercera **capa densa**, se integra una **capa de Dropout** con una tasa aproximada de 0.185 para reducir el sobreajuste.
- La cuarta **capa densa** presenta 51 unidades, activación ReLU, inicializador de pesos `random_normal` y regularización L1.
- Finalmente, se añade la **capa de salida** con 7 neuronas para realizar la predicción final.

Mientras que la arquitectura de la red neuronal para procesar trayectorias, optimizada con Optuna para 100 evaluaciones, se compone de las siguientes capas y configuraciones:

- Se comienza **inicializando el modelo** con la clase `Sequential` de Keras.
- Se introduce una **capa de aplanamiento** (`Flatten`) utilizando una forma de entrada de $10 \times 2 \times 100$.
- La primera **capa densa** se configura con 82 unidades, función de activación ReLU, inicializador de pesos `glorot_uniform` y regularización L2.
- Inmediatamente después de la primera capa densa, se añade una **capa de normalización por lotes** (`BatchNormalization`) para facilitar una mayor estabilidad durante el entrenamiento.
- La segunda **capa densa** cuenta con 83 unidades, activación ReLU, inicializador de pesos `random_normal` y regularización L2.
- Tras la segunda capa densa, se añade otra **capa de normalización por lotes** (`BatchNormalization`) seguida por una **capa de Dropout** con una tasa de 0.133 para mitigar el sobreajuste.
- La tercera **capa densa** incluye 118 unidades, utiliza la activación tanh, el inicializador `he_normal` y la regularización L1.
- Posteriormente, se incorpora una **capa de BatchNormalization**.
- Para la cuarta **capa densa**, se definen 36 unidades con activación ReLU, inicializador de pesos `random_normal` y regularización L2.

- Se sigue con una **capa de BatchNormalization**.
- La quinta **capa densa** se compone de 30 unidades con activación sigmoide, inicializador `glorot_uniform` y regularización L2.
- La capa densa anterior es complementada con una **capa de BatchNormalization**.
- La sexta **capa densa** está formada por 89 unidades, activación ReLU, inicializador de pesos `random_normal` y regularización L2.
- Finalmente, se establece la **capa de salida** con con 7 neuronas que realizan las predicciones finales.

Ahora, para la base de datos de 50 evaluaciones temporales tenemos:

- Se **inicializa el modelo** utilizando la clase `Sequential` de Keras.
- Se introduce una **capa de aplanamiento** (`Flatten`) con una forma de entrada de $10 \times 2 \times 50$.
- La primera **capa densa** se configura con 82 unidades, activación ReLU, inicializador de pesos `glorot_uniform` y regularización L2.
- La segunda **capa densa** incluye 125 unidades, activación sigmoid, inicializador de pesos `random_normal` y regularización L1.
- A continuación, se añade una **capa de normalización por lotes** (`BatchNormalization`) para estabilizar el aprendizaje.
- La tercera **capa densa** consta de 42 unidades, activación tanh, inicializador de pesos `random_normal` y regularización L1.
- Seguidamente, se inserta una **capa de Dropout** con una tasa de de 0.108.
- Finalmente, se añade la **capa de salida** de 7 unidades.

Es importante señalar que la configuración de la arquitectura de la red no seguía un patrón preestablecido; más bien, fue el resultado de un proceso de optimización empírica. Los esquemas estructurales presentados se derivaron como los más efectivos tras una serie de iteraciones de entrenamiento conducidas por Optuna. Estas configuraciones resultaron ser las que generaron los mejores resultados en términos de rendimiento del modelo durante la fase de entrenamiento.

2.4.2. Campos vectoriales

Sin Optimización de Optuna

Para los campos vectoriales la red sigue estando construida con el módulo Sequential de Tensorflow. Para el caso de las 100 evaluaciones sin Optuna la red está estructurada de la siguiente forma:

- Comienza con una **capa de aplanamiento** (`Flatten`), que transforma el tensor de entrada de dimensiones $2 \times 100 \times 100$ en un vector unidimensional.
- Seguido por una **capa de normalización por lotes** (`BatchNormalization`), que normaliza las entradas de la red para facilitar un entrenamiento más estable y eficiente.
- Se añaden sucesivas **capas densas** (`Dense`) con 100 unidades cada una y función de activación 'relu' para las dos primeras capas, seguidas por capas con 50 y 25 unidades, también con activación 'relu'.

- Intercalada entre las capas densas, se incorpora una **capa de Dropout** con una tasa de 0.2.
- Otra **capa de normalización por lotes** es agregada posterior al Dropout.
- Finaliza con una **capa de salida** (`Dense`) con 7 unidades, encargada de generar las predicciones finales del modelo.

Para el tratamiento de los conjuntos de datos correspondientes a campos vectoriales con 50 puntos por componente, se optó por preservar la arquitectura general de la red previamente definida. No obstante, se realizó un ajuste en el número de neuronas en las capas para alinearse con la cantidad de puntos en el espacio del campo vectorial. Las capas, por tanto, fueron configuradas con 50, 50, 25 y 12 neuronas respectivamente, permitiendo una modelación más precisa dada la densidad de información del campo vectorial.

En el caso de los campos vectoriales definidos por 20 puntos por componente, se procedió a una reducción adicional en el número de neuronas para cada capa, estableciendo cantidades de 20, 20, 10 y 5 neuronas respectivamente. Este enfoque busca prevenir la sobreparametrización ante un conjunto de datos con menos puntos vectoriales, favoreciendo la eficiencia en el cálculo y preservando la capacidad de generalización del modelo.

Con Optimización de Optuna

En el análisis del campo vectorial basado en 100 evaluaciones, la arquitectura de la red se ha diseñado de acuerdo con las siguientes especificaciones:

- Se **inicializa el modelo** utilizando la clase `Sequential` de Keras, preparando la red para la adición secuencial de capas.
- Una **capa de aplanamiento** (`Flatten`) transforma el tensor de entrada de dimensiones $2, 100, 100$ en un vector unidimensional.
- Se introduce una **capa de normalización por lotes** (`BatchNormalization`) para estandarizar las entradas antes de pasar a la primera capa densa.
- La **primera capa densa** consta de 118 unidades con función de activación sigmoide, favoreciendo la capacidad del modelo para manejar relaciones no lineales. La inicialización de los pesos se realiza con `glorot_uniform`, y se aplica una regularización L1 con un valor de 0.001.
- Se añade una capa **Dropout** con una tasa de 0.3177 tras la primera capa densa.
- Una segunda **capa de normalización por lotes** se coloca antes de la siguiente capa densa.
- La **segunda capa densa** se compone de 114 unidades y también utiliza la activación sigmoide. Esta capa se inicializa con `random_normal` y se le aplica igualmente una regularización L1.
- Finalmente, se añade la **capa de salida** con 7 unidades.

Para el procesamiento de 50 evaluaciones del campo vectorial, se adopta una arquitectura de red neuronal con la configuración detallada a continuación:

- Iniciamos con la **construcción del modelo**, utilizando la clase `Sequential`.
- Introducimos una **capa de aplanamiento** (`Flatten`) para convertir la entrada tridimensional $(2, 50, 50)$ en un vector unidimensional.

- Establecemos la **primera capa densa** con 16 unidades, optando por la función de activación 'relu', una inicialización de pesos mediante `random_normal`, y aplicamos regularización L1.

- Configuramos la **segunda capa densa** con 68 unidades, manteniendo la activación 'relu' e inicialización de pesos `random_normal`, y continuamos con la regularización L1.

- La **tercera capa densa** se conforma con 106 unidades, incorporando nuevamente la activación 'relu', pero cambiando la inicialización de pesos a `he_normal` con regularización L1.

- Añadimos la **cuarta capa densa** que cuenta con 124 unidades y sigue utilizando la activación 'relu'. Aquí retornamos a la inicialización `random_normal` y optamos por una regularización L2.

- Además, implementamos una capa de **Dropout** con una tasa de 0.157.

- Concluimos con la **capa de salida**, que consta de 7 unidades.

Para la base de datos más pequeña, que consiste en 20 evaluaciones del campo vectorial, la herramienta de optimización de hiperparámetros Optuna ha seleccionado un modelo con la siguiente configuración de red neuronal:

- Se **inicia el modelo** con la implementación de la clase `Sequential`.

- Una **capa de aplanamiento** (`Flatten`) prepara los datos transformando el tensor de entrada con dimensiones $2 \times 50 \times 50$ en un vector unidimensional.

- La **primera capa densa** contiene 16 unidades, utiliza una función de activación 'relu', inicializa los pesos con una distribución `random_normal` y aplica una regularización L1.

- Seguidamente, la **segunda capa densa** incrementa el número de unidades a 68, manteniendo la función de activación 'relu' y la misma configuración de inicialización y regularización que la primera capa.

- La **tercera capa densa** amplía aún más el número de unidades a 106, con la función de activación 'relu'. Esta capa se distingue por la inicialización de pesos `he_normal` y sigue aplicando regularización L1.

- En la **cuarta capa densa**, se establecen 124 unidades con activación 'relu', inicialización `random_normal` y, en esta ocasión, se introduce una regularización L2.

- A continuación, se incluye una capa de **Dropout** con una tasa de 0.157 para reducir el sobreajuste.

- Finalmente, se añade la **capa de salida** con 7 unidades.

Estas configuraciones de red neuronal son el resultado de un enfoque iterativo y experimental que busca el equilibrio entre complejidad y rendimiento computacional. La diversidad en las estructuras de red subraya la adaptabilidad de la metodología de aprendizaje automático para ajustarse a diferentes escalas de datos, demostrando la flexibilidad de las redes neuronales para ajustarse a los requisitos específicos de cada conjunto de datos de campo vectorial.

2.5. Red neuronal convolucional

En lo que respecta a las redes convolucionales, se establecieron estructuras variadas, cada una meticulosamente adaptada a las particularidades de los conjuntos de datos, distinguiendo entre la información derivada de trayectorias y la proveniente de campos vectoriales, para la configuración inicial sin la optimización por Optuna.

Con la integración de Optuna en nuestro flujo de trabajo, se procedió a desarrollar una metodología de búsqueda de hiperparámetros altamente detallada, cuyo objetivo era refinar y mejorar las redes neuronales convolucionales. Este sistema de optimización no estaba vinculado a una tipología de datos específica, sino que buscaba la mejora general del modelo. A continuación se expondrá el esquema de búsqueda de hiperparámetros implementado para las redes convolucionales en el entorno optimizado por Optuna:

1. **Definición de la Función Objetivo:** Se establece la función 'objective' que Optuna minimizará mediante el ajuste de hiperparámetros y la validación de la red neuronal.

2. **Inicialización del Modelo Sequential:** Se crea un modelo secuencial, utilizando la clase 'Sequential' de Keras, que permitirá apilar linealmente las capas de la red.

3. **Configuración de Capas Convolucionales:** Dentro de un bucle, se configuran dinámicamente varias capas convolucionales, donde para cada una se define:

   - **Número de Capas Convolucionales**: Se prueban configuraciones con 1 a 3 capas convolucionales.
   - **Número de Filtros**: En cada capa convolucional, el número de filtros varía entre 8 y 64.
   - **Tamaño de los Filtros**: Se sugieren tamaños de filtro desde 2×2 hasta 5×5.
   - **Funciones de Activación**: Para las capas convolucionales y densas, se consideran las funciones de activación 'relu', 'tanh' y 'sigmoid'.
   - **Inicializadores del Kernel**: Se exploran 'HeNormal', 'glorot_uniform' y 'random_normal' como métodos de inicialización de pesos.
   - **Regularizadores**: Se evalúan regularizadores tanto L1 como L2, con un coeficiente de 0.001 para L1 y valores predefinidos para L2.

4. **Capa de Aplanamiento:** Se introduce una capa 'Flatten' para convertir los mapas de características tridimensionales en un vector unidimensional.

5. **Añadido de Capas Densas:** Se añade una serie de capas densas de 1 a 4, con Optuna sugiriendo el número de neuronas, la función de activación, la inicialización del kernel y la regularización para cada una.

   - **Tasa de Dropout**: Después de las capas densas, se consideran tasas de Dropout desde 0.1 hasta 0.5.
   - **Número de Unidades en Capas Densas**: Se prueban cantidades de unidades en capas densas que varían entre 16 y 128.
   - **Inclusión de BatchNormalization**: Se decide sobre la inclusión de capas de normalización por lotes después de cada capa densa.

6. **Capa de Salida:** Se añade la capa de salida con una función de activación personalizada para la predicción de los valores objetivo.

7. **Compilación del Modelo:** Se elige el optimizador Adam y se establece la función de pérdida como '*mean_squared_error*'. Luego se procede al entrenamiento del modelo con los datos proporcionados.

8. **Optimización Mediante Optuna:** Se inicia la optimización con Optuna, ejecutando una serie de pruebas para determinar la mejor configuración de hiperparámetros.

9. **Resultados de la Optimización:** Tras finalizar la optimización, se imprimen los detalles del mejor ensayo, incluyendo el valor de la función objetivo y los hiperparámetros resultantes.

La optimización de hiperparámetros para las redes convolucionales mediante Optuna presenta desafíos únicos en comparación con las redes densamente conectadas. Una consideración crítica en la configuración de una red convolucional es el mantenimiento de dimensiones apropiadas de los datos a través de las capas de la red. A diferencia de las capas densas, las capas convolucionales y de agrupación (pooling) alteran el tamaño del volumen de entrada, lo que requiere un cálculo cuidadoso para asegurar que las dimensiones sean compatibles en cada paso de la secuencia de la red.

1. **Determinación Dinámica de Dimensiones:** Optuna debe considerar las dimensiones de los datos de entrada y cómo estas cambian después de cada capa convolucional y de agrupación para evitar errores de incompatibilidad de dimensiones. Esto implica un proceso iterativo de ajuste de hiperparámetros que no solo optimiza el rendimiento sino que también garantiza la coherencia dimensional.

2. **Validación de Dimensiones Tras Convoluciones:** Cada capa convolucional aplicada reduce el tamaño espacial del volumen de entrada dependiendo del número y tamaño de los filtros. La función de optimización debe, por lo tanto, validar que las dimensiones resultantes sean suficientes para aplicar las siguientes capas convolucionales o de agrupación.

3. **Incorporación de Capas de Agrupación:** La decisión de incluir capas de agrupación (MaxPooling) debe ser tomada teniendo en cuenta cómo la reducción de dimensiones impactará las capas subsiguientes, un aspecto que Optuna maneja mediante la sugerencia condicional basada en las dimensiones actuales del volumen de entrada.

Por ende, la optimización con Optuna para redes convolucionales requiere una estrategia sofisticada que pueda navegar por el espacio de hiperparámetros multidimensional, evaluando no solo la eficacia predictiva sino también la viabilidad estructural de la arquitectura propuesta. Esto subraya la complejidad inherente al diseño de redes convolucionales donde cada capa y cada hiperparámetro pueden tener implicaciones significativas en la conformación final del modelo.

2.5.1. Trayectorias

Sin Optimización de Optuna

La estructura de la red para los datos de 200 evaluaciones en el tiempo tiene la siguiente forma:

1. La **dimensión de entrada** se define como $(200, 10, 2)$, estableciendo la forma de los datos que se procesarán a través de la red.

2. Incorporamos una **capa convolucional 2D (Conv2D)** con 200 filtros de 5×1, inicializados utilizando la técnica de He normal y con una regularización L1 con coeficiente de 0.001. Se selecciona 'tanh' como función de activación, resultando en una dimensión de salida de $(196, 10, 50)$ dado que no se aplica padding.

3. A continuación, una **capa de MaxPooling2D** con un tamaño de ventana de 2×1 reduce la dimensionalidad espacial, proporcionando una salida de $(98, 10, 50)$.

4. Otra **capa Conv2D** sigue, con 49 filtros de 5×1, manteniendo las mismas configuraciones de inicialización y regularización. La salida de esta capa se calcula como $(94, 10, 49)$.

5. La siguiente **capa de MaxPooling2D**, de idéntico tamaño de ventana, reduce aún más las dimensiones a $(47, 10, 49)$.

6. Integramos una tercera **capa Conv2D** con 24 filtros, la cual ajusta las dimensiones de salida a $(43, 10, 24)$.

7. Una adicional **capa de MaxPooling2D** sigue, dando como resultado una dimensión de $(21, 10, 24)$.

8. La cuarta **capa Conv2D** emplea 50 filtros de 5×2, y la salida se ajusta a $(17, 9, 50)$.

9. La última **capa de MaxPooling2D** continúa el proceso de reducción, llevando las dimensiones a $(8, 9, 50)$.

10. Utilizamos una **capa Flatten** para transformar la salida multidimensional en un vector unidimensional de 3600 elementos.

11. Se introduce una **capa densa** con 200 nodos, empleando la función de activación 'tanh', inicialización He normal, y regularización L1.

12. Procedemos con otra **capa densa** que consta de 100 nodos, también con 'tanh' como función de activación y regularización L1.

13. La **tercera capa densa** se compone de 50 nodos y utiliza la función de activación 'ReLU'.

14. Una **capa de Dropout** se añade con una tasa de 0.3, desconectando aleatoriamente el 30 % de los nodos durante el entrenamiento.

15. La construcción finaliza con la **capa de salida**, una capa densa con 7 nodos.

En términos de estructura, se emplearon capas de MaxPooling para reducir la dimensionalidad de los datos. Estas capas lograron preservar la información más importante, los valores máximos, al tiempo que descartaron información redundante o menos útil. Esta estrategia mejoró la eficiencia computacional de la red y ayudó a prevenir el sobreajuste.

Las capas convolucionales se seleccionaron y ajustaron de acuerdo con las características específicas de los datos. Los detalles sobre la cantidad y los tamaños de los filtros se definieron en función de estos factores. En las primeras capas, con un número más elevado de filtros, la red puede aprender a reconocer una gran cantidad de características simples y distintas. A medida que avanzamos a través de las capas, el número de filtros disminuye, lo que puede ayudar a la red a concentrarse en las características más importantes y combinaciones de estas. Por último, el número de filtros aumenta de nuevo en la última capa convolucional, permitiendo a la red aprender características más complejas y combinaciones de las características aprendidas anteriormente.

Para fomentar aún más la robustez del modelo y prevenir el sobreajuste, se añadió una capa de Dropout. Esta técnica de regularización funciona desactivando aleatoriamente algunas neuronas durante el entrenamiento, forzando a la red a aprender representaciones más sólidas y menos dependientes de cualquier neurona individual.

En la configuración de la red neuronal para procesar 100 evaluaciones temporales, se realizaron ajustes en la cantidad de filtros y la forma de entrada para adaptarse a la disminución en la cantidad de datos. La forma de entrada específica se modificó a $(100, 10, 2)$.

- La **primera capa convolucional** se adapta a la nueva dimensión de los datos reduciendo el número de filtros a 100.
- La **segunda capa convolucional** reduce aún más el número de filtros a 24, reflejando la necesidad de menos parámetros en respuesta a la entrada más pequeña.
- Para la **tercera capa convolucional**, se seleccionan 11 filtros, continuando con la tendencia de disminuir el número de filtros para mantener la eficiencia computacional.
- La **cuarta capa convolucional** mantiene la cantidad de 50 filtros, igualando la configuración utilizada para la red con 200 evaluaciones.

En cuanto a las capas densas posteriores a las convolucionales, se configuran con 100, 50 y 20 nodos respectivamente, siguiendo una lógica similar de reducción proporcional para alinear con la entrada de menor tamaño.

Para adaptar la red neuronal a la base de datos de menor tamaño, se conservó la estructura general mientras se ajustaba el número de filtros en las capas convolucionales y se modificaban las capas densas para alinearlas con el volumen reducido de datos. Los ajustes son los siguientes:

- La **primera capa convolucional** se configura con 50 filtros, adecuados para la captura de características iniciales a pesar de la disminución en la cantidad de datos.
- En la **segunda capa convolucional**, el número de filtros se reduce a 12, lo que permite un procesamiento eficiente sin sacrificar la capacidad de extracción de características.
- La **tercera capa convolucional** se ajusta a 9 filtros, y en este punto se decide omitir la operación de MaxPooling que normalmente seguiría, debido a las restricciones de tamaño de los datos de entrada.
- Manteniendo la consistencia con la primera capa, la **cuarta capa convolucional** se establece con 50 filtros.
- Para las **capas densas**, se eligen configuraciones de 50, 20 y 20 nodos respectivamente, proporcionando un procesamiento y una integración de características adecuados para la escala de los datos procesados.

Esta estructura detallada refleja un enfoque cuidadoso para mantener la integridad del procesamiento de características a pesar de la reducción en la dimensión de los datos, garantizando que la red permanezca competente para realizar predicciones precisas.

Con Optimización de Optuna

2.5.2. Campos vectoriales

Sin Optimización de Optuna

Los campos vectoriales siguen teniendo la misma estructura, sin embargo cambia de nuevo el número de filtros y el tamaño del pooling. Para el caso de los campos vectoriales de 100 evaluaciones tenemos:

1. La **capa convolucional 2D inicial** cuenta con 50 filtros de 5×5, activación 'tanh', inicializador HeNormal y regularización L1 de 0.001. Esta capa produce una salida con dimensiones de (96, 96, 50) tras la convolución.
2. La subsiguiente **capa de MaxPooling2D** con ventana de 2×2 reduce el tamaño de la salida a (48, 48, 50).

3. Una segunda **capa convolucional 2D** con 24 filtros de 5×5 mantiene la misma configuración de activación y regularización, generando una salida de (44, 44, 24).
4. Otra **capa de MaxPooling2D**, idéntica a la anterior, disminuye la salida a (22, 22, 24).
5. La tercera **capa convolucional 2D** utiliza 11 filtros de 5×5, produciendo una salida de (18, 18, 11).
6. Una tercera **capa de MaxPooling2D** prosigue con la reducción de dimensiones a (9, 9, 11).
7. La cuarta **capa convolucional 2D** aplica 50 filtros de 5×5, y la salida resultante es de (5, 5, 50).
8. La final **capa de MaxPooling2D** lleva las dimensiones a (2, 2, 50).
9. Una **capa Flatten** transforma la salida multidimensional en un vector unidimensional de 200 elementos ($2 \times 2 \times 50$).
10. Se introduce una **capa densa** con 200 nodos, activación 'tanh', inicializador HeNormal y regularización L1, manteniendo un tamaño de salida de 200.
11. La siguiente **capa densa** con 100 nodos también utiliza 'tanh', HeNormal y L1, con una salida de 100.
12. La tercera **capa densa** dispone de 50 nodos y activación 'relu', con una salida de 50.
13. Una **capa de Dropout** con una tasa del 30 % se emplea para mitigar el sobreajuste, sin cambiar el tamaño de salida de la capa previa.
14. Finalmente, la **capa de salida** con 7 nodos completa la red.

Para el conjunto de datos correspondiente a 50 evaluaciones temporales, la estructura de la red se ajusta principalmente en la cantidad de filtros y nodos en las capas convolucionales y densas respectivamente. Se detallan los cambios:

- La **primera capa convolucional** mantiene un número de 50 filtros por lo que la salida queda como (46,46,50). Al añadir la capa de Maxpooling la salida queda de tamaño (23,23,50).
- La **segunda capa convolucional** reduce el número de filtros a 11. La salida es del tamaño (19,19,11). Tras la capa de Maxpooling, la salida es de tamaño (9,9,11)
- En la **tercera capa convolucional**, se utilizan 9 filtros, ajustándose a la disminución progresiva de la dimensionalidad de los datos. La salida queda de tamaño (5,5,9). Aquí se deja de aplicar la capa de Maxpooling por el tamaño de los datos.
- La **cuarta capa convolucional** vuelve a emplear 50 filtros, manteniendo la consistencia en la capacidad de extracción de la red. La salida queda de tamaño (1,1,50).
- Las **capas densas** se configuran con 200, 100 y 50 nodos respectivamente, lo que refleja un diseño escalonado para la integración gradual de las características aprendidas y la toma de decisiones finales.

Y para las 20 evaluaciones se redujeron aún más el número de capas. La estructura queda de la siguiente forma:

- **Capa convolucional**: 32 filtros de 3×3, activación 'tanh', salida $18 \times 18 \times 32$.
- **MaxPooling2D**: ventana 2×2, salida $9 \times 9 \times 32$.

- **Capa convolucional**: 64 filtros de 3×3, salida $7 \times 7 \times 64$.
- **MaxPooling2D**: salida $3 \times 3 \times 64$.
- **Flatten**: convierte la salida a un vector unidimensional.
- **Capas densas**: secuencia de capas con 128, 64 y 64 nodos respectivamente, alternando activaciones 'tanh' y 'relu'.
- **Dropout**: aplicado dos veces con una tasa del 30 % para reducir el sobreajuste.
- **Capa de salida**: 7 nodos

Con esta estructura, la red apunta a una eficiente extracción de características y adecuada generalización para la clasificación o regresión de datos.

Con Optimización de Optuna

Con Optuna, las redes tomaron estructura completamente diferentes. El modelo de red neuronal, definido para procesar datos de entrada de (100, 100, 2), se compone de la siguiente manera:

- La **primera capa convolucional** contiene 16 filtros de 4×5, usa activación 'relu', inicialización $'random\_normal'$, y regularización L2. La salida de esta capa, sin padding, es de (97, 96, 16).
- La **segunda capa convolucional** aumenta a 42 filtros de 2×4, con activación 'tanh', y las mismas especificaciones de inicialización y regularización. Después de esta capa y el MaxPooling de 2×2, la salida se reduce a (48, 46, 42).
- La **tercera capa convolucional** utiliza 51 filtros de 3×4, continúa con activación 'tanh' pero cambia la inicialización a 'HeNormal'. Incluye MaxPooling de 2×2, dejando una salida de $23 \times 21 \times 51$.
- La **capa Flatten** convierte la salida tridimensional en un vector unidimensional, permitiendo la transición a la siguiente capa densa.
- La **primera capa densa** tiene 108 nodos con activación 'tanh', inicialización 'HeNormal', y regularización L2.
- La **segunda capa densa** sigue con 105 nodos, usa activación 'relu', inicialización $'glorot\_uniform'$, y regularización L1.
- La **capa de salida** cuenta con 7 nodos para la tarea específica del modelo.

Para el caso de la base de datos de tamaño (50,50,2), tenemos que la red es:

- La **primera capa convolucional** cuenta con 16 filtros de 4×5, activación 'relu', inicialización 'random_normal', y una regularización L2. La dimensión de salida es de (47, 46, 16).
- La **segunda capa convolucional** emplea 42 filtros de 2×4, activación 'tanh', con la misma inicialización y regularización. Tras esta capa y un MaxPooling de 2×2, la salida se reduce a (23, 21, 42).
- Una **tercera capa convolucional** utiliza 51 filtros de 3×4, activación 'tanh', inicialización 'HeNormal', y regularización L2. Después del MaxPooling de 2×2, las dimensiones de salida se convierten en (10, 9, 51).

- Una **capa Flatten** transforma la salida de la última capa convolucional en un vector unidimensional para facilitar la transición a las capas densas.
- La **primera capa densa** tiene 108 nodos con activación 'tanh', inicialización 'HeNormal', y regularización L2, promoviendo la extracción de características complejas.
- La **segunda capa densa** sigue con 105 nodos, activación 'relu', inicialización 'glorot_uniform', y regularización L1.
- Finalmente, la **capa de salida** contiene 7 nodos.

Para finalizar, para los campos vectoriales con tamaño (20, 20, 2), la estructura de la red es:

- Se comienza con una **capa convolucional** dotada de 58 filtros de 4×1, utilizando la función de activación 'relu'. La inicialización del kernel se hace con una distribución normal aleatoria y se añade una regularización L1. Asumiendo que no hay padding aplicado, la dimensión de salida de esta capa es de (17, 20, 58).
- Una **capa de MaxPooling2D** con un tamaño de ventana de 2×2 reduce las dimensiones espaciales a (8, 10, 58).
- A continuación, la **capa Flatten** convierte la matriz tridimensional resultante en un vector unidimensional.
- La **primera capa densa** incorpora 28 nodos y emplea una función de activación 'relu', con inicialización 'HeNormal' y regularización L1.
- La **segunda capa densa** aumenta el número de nodos a 41, manteniendo la misma configuración de activación, inicialización y regularización.
- La **tercera capa densa** amplía aún más la capacidad del modelo con 120 nodos, siguiendo el mismo patrón de activación y regularización.
- Se introduce una **capa de Dropout** con una tasa del 30 %.

Las redes convolucionales descritas muestran una adaptación cuidadosa a diferentes tamaños de entrada, reflejando una metodología deliberada y estratégica para la extracción de características relevantes de los datos. Las configuraciones variadas de capas convolucionales, densas y de regularización indican una búsqueda equilibrada entre la captura de la complejidad inherente a los datos y la prevención de sobreajuste.

2.6. Red Neuronal Densa Función de Costo Personalizada

La selección de la función de costo en el entrenamiento de redes neuronales, incluyendo las convolucionales y densas, es un aspecto crítico que impacta directamente en la eficacia y exactitud del modelo. Hemos elegido el Error Cuadrático Medio (MSE) como nuestra principal función de costo. Este criterio, que mide la discrepancia entre las predicciones de la red y los valores reales, se define como la suma de las diferencias al cuadrado entre estos valores, dividida por el número total de elementos. Esta métrica es ampliamente reconocida por su capacidad para proporcionar una evaluación clara y cuantitativa de la exactitud de las predicciones de un modelo.

La aplicación del MSE en nuestro modelo, que intenta capturar la dinámica de un sistema de ecuaciones diferenciales que modela un Toggle Switch, presenta desafíos únicos. Dada la naturaleza del sistema, donde pequeñas variaciones en los coeficientes pueden traducirse en cambios significativos en la dinámica del modelo, el MSE ofrece una manera de medir con precisión estas

diferencias. Sin embargo, un aspecto crítico a considerar es cómo el MSE, en su forma estándar, podría no reflejar completamente la complejidad de la relación entre los parámetros del sistema y su comportamiento dinámico.

Consideremos, por ejemplo, un conjunto de datos experimentales representados por la Ecuación [2.1], con coeficientes conocidos $a_1 = 1$, $a_2 = 7$, $b_1 = 4$, $b_2 = 1$, $d_1 = 4$, $d_2 = 8$, y $n = 3$, como se muestra en la Figura 2.6. Tras entrenar la red, podríamos obtener un conjunto de coeficientes predichos como $a_1 = 2$, $a_2 = 9$, $b_1 = 4$, $b_2 = 3$, $d_1 = 4$, $d_2 = 10$, y $n = 4$, que se ilustran en la Figura 2.7. El cálculo del MSE entre los coeficientes reales y predichos se lleva a cabo de la siguiente manera:

$$MSE = \frac{1}{7}\left((a_{1,\text{real}} - a_{1,\text{pred}})^2 + \cdots + (n_{\text{real}} - n_{\text{pred}})^2\right), \tag{2.7}$$

donde cada término representa la diferencia al cuadrado entre los valores reales y los predichos para cada coeficiente. En nuestro ejemplo, el cálculo sería:

$$\begin{aligned} MSE =& \frac{1}{7}((a_{1,\text{real}} - a_{1,\text{pred}})^2 + (a_{2,\text{real}} - a_{2,\text{pred}})^2 + (b_{1,\text{real}} - b_{1,\text{pred}})^2 + (b_{2,\text{real}} - b_{2,\text{pred}})^2 \\ &+ (d_{1,\text{real}} - d_{1,\text{pred}})^2 + (d_{2,\text{real}} - d_{2,\text{pred}})^2 + (n_{\text{real}} - n_{\text{pred}})^2) \\ &= \frac{1}{7}((1-2)^2 + (7-9)^2 + (4-4)^2 + (1-3)^2 + (4-4)^2 + (8-10)^2 + (3-4)^2) \\ &= \frac{1}{7}(1+4+0+4+0+4+1) \\ &= \frac{1}{7} * 14 \\ &= 2 \end{aligned} \tag{2.8}$$

Este valor, que se busca minimizar, refleja la diferencia entre los parámetros reales y los predichos. Sin embargo, esta función de costo no incorpora información específica del problema que estamos abordando, intentando "a fuerza bruta" minimizar la discrepancia entre los parámetros.

Para superar esta limitación, hemos propuesto un enfoque que incluye la información intrínseca de los datos. Consiste en minimizar las diferencias entre los campos vectoriales generados por los datos reales y los predichos. Esto se logra comparando cada una de las flechas en la Figura 2.6 con las correspondientes en la Figura 2.7. En la Figura 2.8 se presentan ambos campos superpuestos, y en la Figura 2.9 se muestra la diferencia entre ellos. Este método permite que la red no solo se aproxime a los valores de los coeficientes, sino que también capture la estructura subyacente del sistema dinámico representado por los datos.

Esta metodología nos permite no solo aproximar los valores de los coeficientes, sino también capturar la estructura subyacente y la dinámica del sistema dinámico que los datos representan. Al incorporar la totalidad del campo vectorial en el entrenamiento de la red, abordamos la complejidad inherente del sistema biológico en estudio, buscando no solo predecir con precisión los coeficientes, sino replicar fielmente la dinámica del sistema. Con este enfoque, aspiramos a mejorar significativamente la precisión y la relevancia de las predicciones del modelo, alineándolas más estrechamente con las particularidades del sistema biológico investigado.

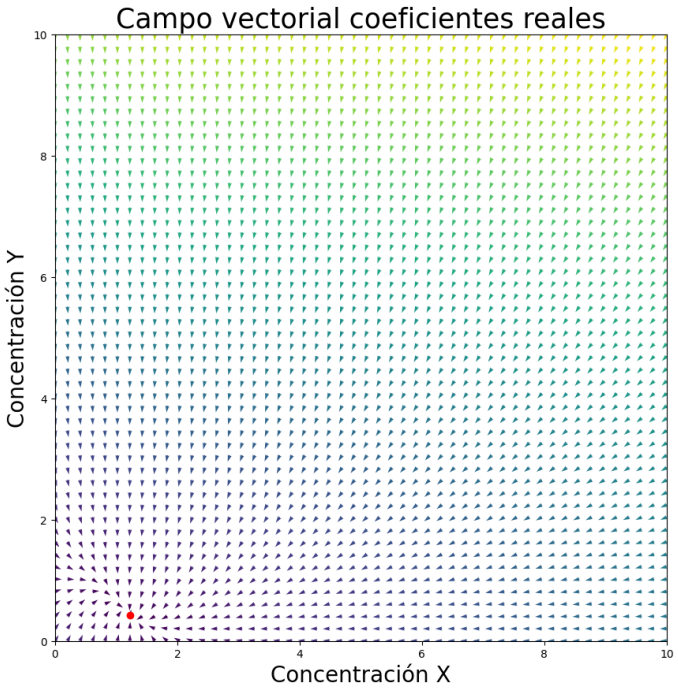


Figura 2.6: Espacio vectorial generado con los coeficientes $a_1 = 1, a_2 = 7, b_1 = 4, b_2 = 1, d_1 = 4, d_2 = 8$ y $n = 3$.

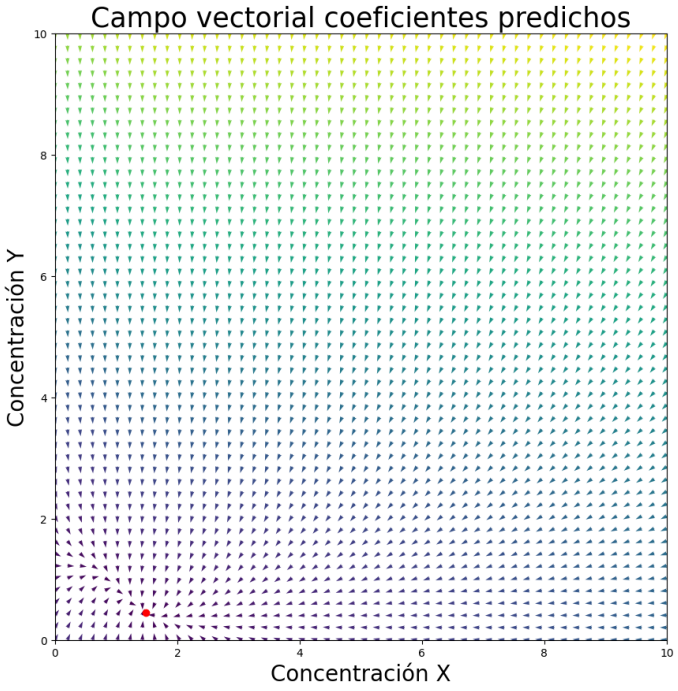


Figura 2.7: Espacio vectorial predicho por la red con los coeficientes $a_1 = 2, a_2 = 9, b_1 = 4, b_2 = 3, d_1 = 4, d_2 = 10$ y $n = 4$.

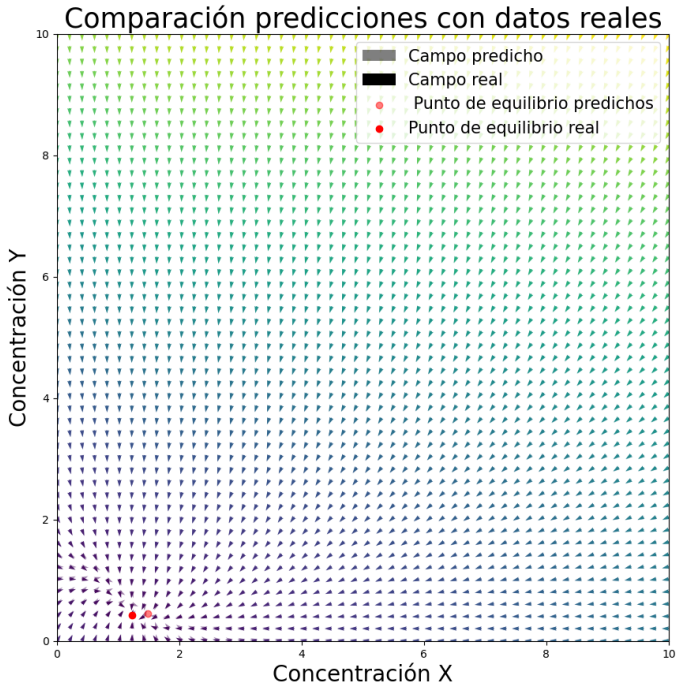


Figura 2.8: Campos vectoriales puestos uno encima del otro para comparar sus diferncias.

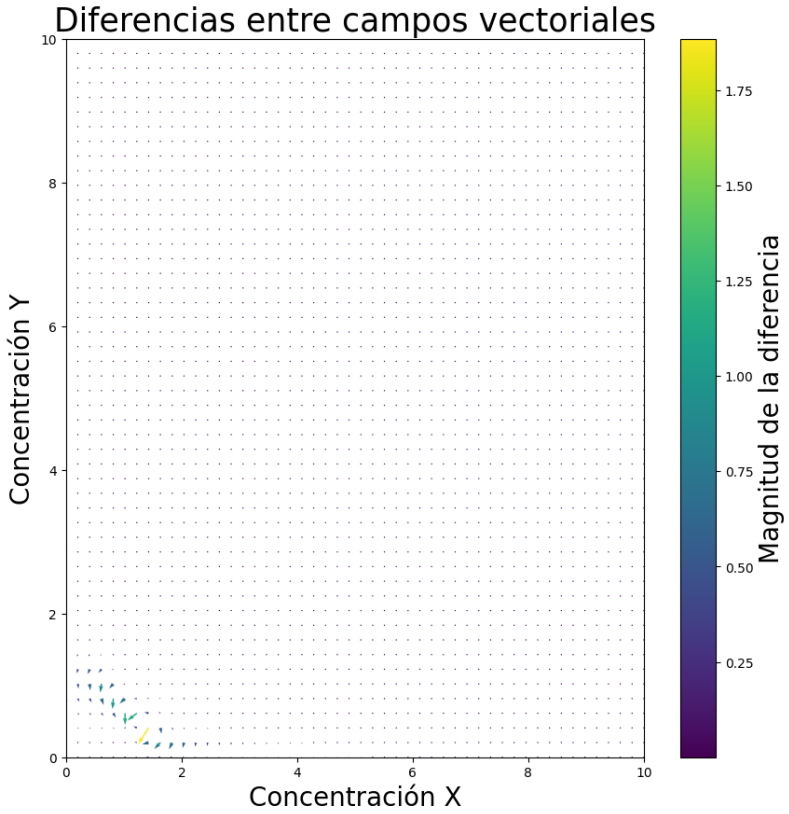


Figura 2.9: Diferencia entre los campos vectoriales reales y predichos.

La forma en que calculamos esta función de costo es la siguiente:

1. **Inicialización y Reshape de Coeficientes:** Se extraen los coeficientes de los valores verdaderos (`y_true`) y los predichos (`y_pred`), y se transforman en tensores de tipo float64. Luego, se redimensionan para tener la forma adecuada para los cálculos siguientes.

2. **Definición de las Ecuaciones Diferenciales (ODEs):** Se definen dos funciones, `f` y `f2`, para los conjuntos de coeficientes reales y predichos, respectivamente. Estas funciones calculan las derivadas `dX_dt` y `dY_dt` para cada conjunto.

3. **Creación de Mallas y Campos Vectoriales:** Se generan mallas bidimensionales usando `tf.meshgrid` y se calculan los campos vectoriales para los coeficientes reales y predichos. Estos campos representan la dirección y magnitud de las derivadas en cada punto de la malla.

4. **Cálculo de la Diferencia entre Campos Vectoriales:** Se calculan las diferencias absolutas entre los campos vectoriales correspondientes y se suman estas diferencias para obtener la magnitud total de la discrepancia entre los dos campos.

5. **Cálculo del Error o Pérdida:** La pérdida se determina como el promedio de la magnitud de la discrepancia entre los campos vectoriales. Esto refleja cuánto difieren los campos generados por los coeficientes predichos de los generados por los coeficientes reales.

Esta función de pérdida personalizada va más allá de minimizar las diferencias entre los coeficientes individuales, enfocándose también en asegurar que los campos vectoriales generados por los coeficientes predichos se asemejen lo más posible a los campos generados por los coeficientes reales, lo cual es crucial en aplicaciones donde la estructura y dinámica del sistema son tan importantes como los valores específicos de los coeficientes.

A partir de aquí se lleva como una red neuronal densamente conectada. Durante la fase de compilación del modelo, se incorpora la función de pérdida personalizada que hemos desarrollado. Esta integración se realiza especificando nuestra función `custom_loss` en el parámetro de pérdida (`loss`) del método de compilación del modelo.

Sin Optimización de Optuna

En el contexto de las redes neuronales sin la aplicación de Optuna, se adopta la misma arquitectura que en las redes densas previamente descritas. La distinción principal reside en el proceso de entrenamiento, donde la función de costo mencionada anteriormente juega un papel fundamental. Esta metodología garantiza una coherencia estructural en las redes neuronales, al tiempo que se explora la eficacia de diferentes funciones de costo en el proceso de aprendizaje.

Con Optimización de Optuna

Con Optuna se hizo la misma búsqueda de hiperparámetros que con las redes densas dando resultados diferentes en la estructura de red. Para la base de datos más grande (2, 100, 100) tenemos:

- Se inicializa con una **capa de aplanamiento (flatten)** que transforma la entrada bidimensional en un vector unidimensional.

- La **primera capa densa** está compuesta por 118 nodos, utiliza la función de activación 'relu'. La inicialización de los pesos se realiza mediante 'HeNormal', y se aplica una regularización L2 para mitigar el sobreajuste. Además, se incorpora una **capa de normalización por lotes (BatchNormalization)** después de esta capa.

- La **segunda capa densa** cuenta con 54 nodos y emplea la función de activación 'tanh'. Al igual que en la primera capa, se usa la inicialización 'HeNormal' y la regularización L2, seguida por otra **capa de BatchNormalization** para mantener la normalización de las entradas en la red.
- La **tercera capa densa** incluye 112 nodos con la función de activación 'relu'. A diferencia de las capas anteriores, utiliza una inicialización $'random\_normal'$ y se aplica una regularización L1.
- Una última **capa de salida densa** que contiene 7 nodos.

La estructura de la red neuronal, optimizada con Optuna para la base de datos de dimensiones (2, 50, 50), se organiza de la siguiente manera:

- La **capa de aplanamiento** que configurada para entradas de tamaño (2, 50, 50).
- Una **primera capa densa** que incluye 118 nodos y emplea la función de activación 'relu'. Se utiliza la inicialización 'HeNormal' y la regularización L2 con un factor de 0.001. Además, se incorpora **BatchNormalization**.
- Seguimos con una **segunda capa densa** compuesta por 54 nodos con función de activación 'tanh', también utiliza la inicialización 'HeNormal' y la regularización L2, seguida de otra capa de **BatchNormalization**.
- La **tercera capa densa** consta de 112 nodos, utiliza 'relu' como función de activación y una inicialización 'random_normal'. Se aplica una regularización L1.
- Finalizamos con la **capa de salida densa** con 7 nodos.

Para la base de datos más chica (2,20,20) tenemos la siguiente forma:

- Se inicia con la **capa de aplanamiento** para aceptar entradas de tamaño (2, 20, 20).
- La **primera capa densa** Consiste en 49 nodos y utiliza la función de activación 'relu'. Los pesos se inicializan mediante el método 'glorot_uniform', y se emplea una regularización L2 con un parámetro de 0.001.
- Seguimos con una **segunda capa densa** que contiene 126 nodos y emplea la función de activación 'sigmoid'. Aquí, la inicialización de los pesos se realiza mediante 'random_normal', y se continúa con la regularización L2. E
- La red finaliza con una **capa de salida** de 7 nodos.

2.6.1. Early Stop

El mecanismo de **Early Stopping** se implementa en el entrenamiento de la red para interrumpir el proceso cuando no se observan mejoras significativas tras un número específico de épocas. Esta técnica no solo previene el sobreentrenamiento, sino que también tiene la capacidad de restaurar los pesos del modelo que lograron las mejores predicciones hasta el momento.

En nuestro estudio, se estableció un parámetro de paciencia de 100 épocas para el Early Stopping. Esto significa que el entrenamiento se detendrá si no se detecta una mejora en la métrica 'val_mean_metric' que es el valor de 'mean_metric' para los datos de validación durante esas 100 épocas consecutivas. Esta configuración es esencial para asegurar que la red alcance un equilibrio óptimo entre el aprendizaje efectivo y la prevención del sobreajuste, ajustando su desempeño basado en la evolución del error en el conjunto de validación.

2.7. Visualización de métricas

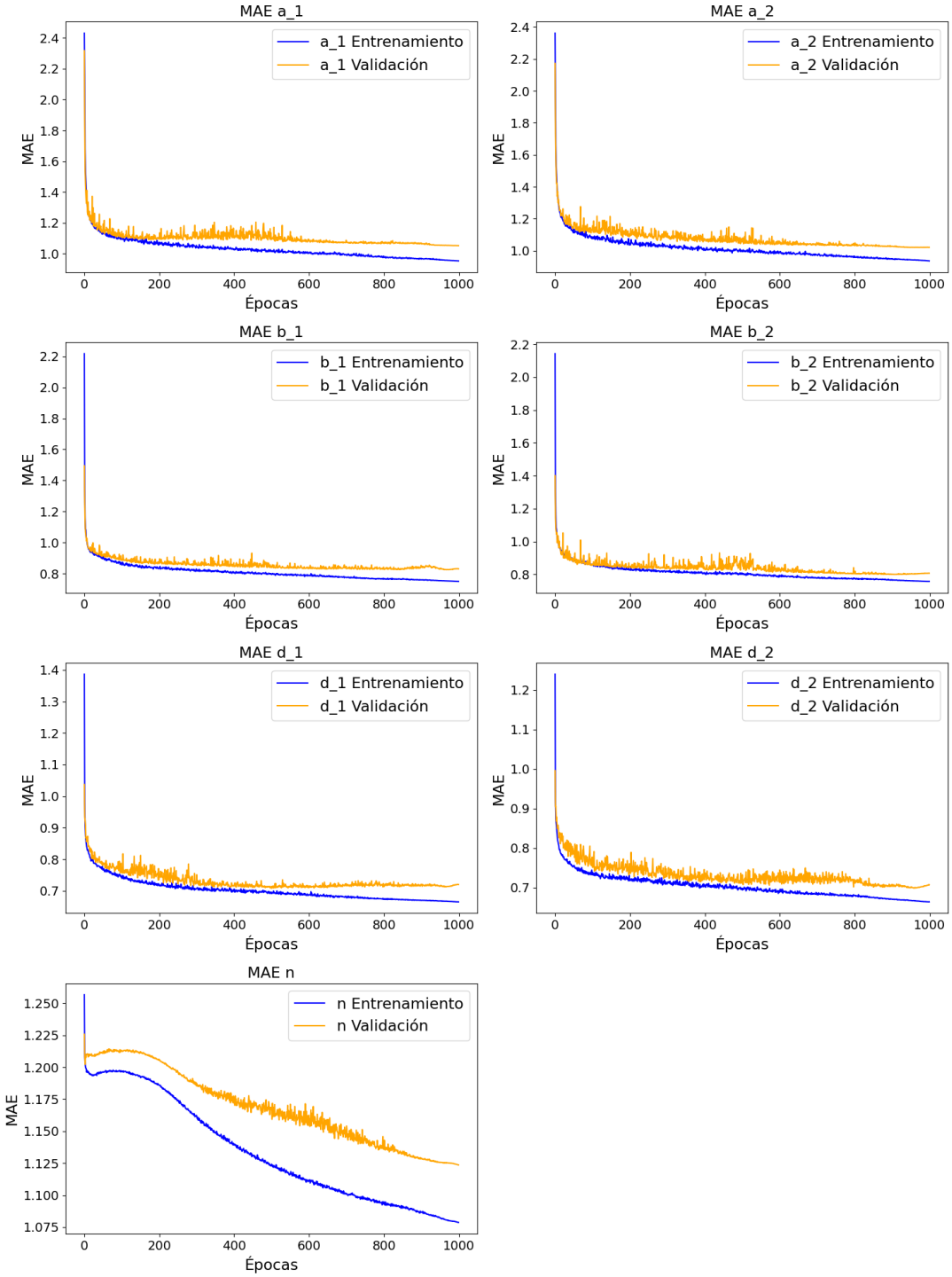


Figura 2.10: Evolución de las Métricas Durante el Entrenamiento del Modelo. Esta figura muestra el comportamiento de las métricas clave, incluyendo 'mean_metric' y los errores absolutos medios individuales para cada coeficiente del modelo, a lo largo de las épocas de entrenamiento.

Tras completar el entrenamiento del modelo, procedemos a realizar un análisis detallado de su rendimiento a lo largo de las épocas mediante la visualización de cada una de las métricas. Este procedimiento es fundamental para identificar posibles casos de sobreajuste y comprender qué coeficientes representan mayores desafíos para el aprendizaje de la red. Estos insights se obtienen a través de gráficos que muestran la evolución de las métricas en cada época, proporcionando una representación clara y comprensible del comportamiento del modelo a lo largo del tiempo. Un ejemplo representativo de este tipo de visualización se encuentra en la Figura 2.10, donde se ilustra la variación de las métricas clave a lo largo de las épocas de entrenamiento.

2.8. Evaluación del Modelo mediante los datos de testeo

En la fase de evaluación de nuestro modelo, se realizan una serie de procedimientos esenciales para determinar su rendimiento y precisión. Estos pasos se describen a continuación:

1. **Evaluación del Modelo**: Utilizando `model.evaluate`, evaluamos el modelo en el conjunto de datos de prueba, obteniendo métricas clave como el Error Absoluto Medio (MAE) para cada coeficiente, la función de costo (MSE) y la métrica que antes definimos *mean_metric*.

2. **Generación de Predicciones**: Aplicamos `model.predict` en el conjunto de prueba para obtener predicciones del modelo, lo que nos permite una comparación directa con los valores reales.

3. **Cálculo del Error Absoluto Medio (MAE)**: Empleamos `mean_absolute_error` de *Scikit-learn* para calcular el MAE, proporcionando una medida precisa de la diferencia media entre los valores predichos y los reales.

4. **Análisis Estadístico del Error**: Calculamos la desviación estándar del error, ofreciendo una medida de la variabilidad de los errores del modelo. Además, identificamos el error absoluto máximo, destacando el caso más extremo de error en las predicciones.

5. **Visualización de la Distribución del Error**: Creamos histogramas utilizando `plt.hist` de *Matplotlib* para visualizar la frecuencia y distribución de los errores.

6. **Evaluación del Error Relativo**: Calculamos el error relativo para cada dato, que es la suma del error absoluto dividida por la suma de los valores reales. Este análisis se complementa con la visualización de la distribución del error relativo a través de histogramas, proporcionando una perspectiva adicional sobre la precisión del modelo en relación con la magnitud de los valores reales.

Estos pasos nos proporcionan una visión integral del rendimiento del modelo, destacando tanto su precisión general como la variabilidad y la naturaleza de los errores cometidos. Este análisis detallado es crucial para validar la eficacia del modelo y para identificar áreas de mejora potencial.

2.9. Guardar resultados

Una vez completado el entrenamiento y evaluación del modelo, procedemos a la fase de almacenamiento de los resultados. Esto implica que los modelos entrenados se almacenan en archivos con la extensión `.h5`, que corresponden al formato Hierarchical Data Format versión 5 (HDF5).

El HDF5 es un formato avanzado diseñado para el almacenamiento eficiente de grandes volúmenes de datos numéricos. Capaz de contener una variedad de tipos de datos, desde imágenes y matrices numéricas hasta tablas complejas, el HDF5 es ampliamente utilizado en campos como la física, ingeniería, astronomía y biología computacional, gracias a su capacidad para manejar

conjuntos de datos extensos y complejos [91].

En el contexto específico de Keras y del aprendizaje automático en general, los archivos `.h5` son particularmente valiosos para almacenar los pesos de los modelos entrenados. Esta práctica ofrece la ventaja de poder cargar posteriormente estos pesos en un modelo para realizar inferencias, sin necesidad de reentrenar, lo que ahorra tiempo y recursos computacionales significativos. Además, el intercambio de estos archivos `.h5` facilita la reproducibilidad de los resultados y promueve la colaboración, permitiendo a otros investigadores y desarrolladores basarse en modelos previamente entrenados para futuras investigaciones o aplicaciones.

2.10. Pruebas gráficas

Tras el almacenamiento del modelo, se procede a su reimportación para una fase de simulación y análisis visual posterior. En esta etapa, generamos nuevos datos de manera aleatoria, imitando así las características de datos obtenidos experimentalmente. Estos datos se visualizan gráficamente para analizar su comportamiento dinámico.

Posteriormente, estos datos simulados se introducen al modelo reimportado para estimar los coeficientes que mejor los representan. Utilizando los coeficientes estimados, reproducimos gráficamente el sistema, lo que nos permite una comparación visual entre los resultados predichos y los esperados, proporcionando una validación directa y efectiva del modelo.

Además de la visualización gráfica, se realiza un análisis de los puntos de equilibrio en los datos simulados y en los predichos por el modelo. Los puntos de equilibrio son estados en los cuales las variables del sistema se estabilizan y dejan de cambiar con el tiempo, siendo cruciales para comprender la dinámica del sistema. Para hallar estos puntos en nuestro estudio, empleamos el método `fsolve` de la biblioteca *SciPy* de Python, que es una herramienta eficaz para resolver sistemas de ecuaciones no lineales.

El proceso se inicia definiendo el sistema de ecuaciones diferenciales que describen nuestro modelo. Luego, se utiliza `fsolve` para encontrar soluciones a este sistema, es decir, los valores de las variables en los cuales las derivadas (cambios) de estas variables se reducen a cero. Esto se logra proporcionando al método `fsolve` una función que representa nuestro sistema de ecuaciones y una estimación inicial de los valores de las variables. En nuestro caso, esta adivinanza inicial se establece como un vector cero, asumiendo que los puntos de equilibrio están cerca del origen.

Una vez que `fsolve` procesa esta información, devuelve una solución que corresponde a los puntos de equilibrio del sistema. Estos puntos son luego analizados y comparados entre los datos reales y los generados por el modelo, proporcionando una comprensión más profunda de la precisión y aplicabilidad de nuestro modelo en situaciones experimentales reales.

La comparación final se realiza mediante la superposición de las gráficas de los datos simulados (experimentales) y los generados por los coeficientes predichos por el modelo. Esta comparativa gráfica permite una evaluación visual directa de la precisión del modelo, destacando las diferencias entre los comportamientos reales y los predichos, y proporcionando una herramienta valiosa para la validación y ajuste del modelo.

En lo que respecta a los campos vectoriales, la comparativa efectuada es análoga a la descrita en la sección dedicada a la Red Neuronal Densa con Función de Costo Personalizada. En este contexto, la Figura 2.6 representa los datos simulados, emulando resultados experimentales, mientras

que la Figura 2.7 ilustra el campo vectorial resultante al aplicar los coeficientes pronosticados por el modelo en el sistema de ecuaciones del Toggle Switch. La visualización de las discrepancias entre estos campos vectoriales se facilita mediante la Figura 2.9, que expone de manera gráfica las diferencias existentes entre los datos experimentales simulados y las predicciones del modelo. Esta comparación gráfica es fundamental, ya que proporciona una perspectiva clara sobre la precisión y efectividad del modelo para replicar y entender el comportamiento del sistema bajo estudio.

En el caso de las Trayectorias, el procedimiento seguido fue similar. Inicialmente, generamos un conjunto de datos sintéticos que representan las trayectorias posibles dentro del sistema. Estos datos fueron luego visualizados gráficamente, como se muestra en la Figura 2.11. Posteriormente, introducimos estos datos sintéticos en nuestro modelo entrenado para obtener las predicciones de los coeficientes correspondientes. Con estos coeficientes pronosticados, procedimos a calcular las trayectorias resultantes mediante su inserción en el sistema de ecuaciones del Toggle Switch. Las trayectorias obtenidas, que reflejan las predicciones del modelo, se presentan en la Figura 2.12. Este enfoque nos permite evaluar la capacidad del modelo para predecir trayectorias que se alineen con los datos experimentales simulados, ofreciendo así una valiosa herramienta para el análisis y validación del modelo.

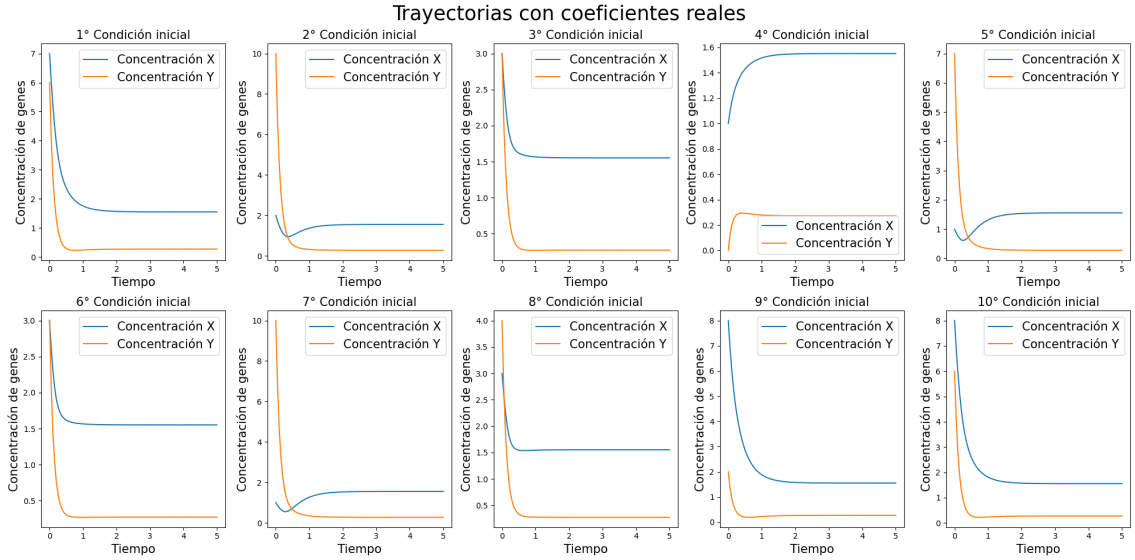


Figura 2.11: Trayectorias de los coeficientes reales. En este caso son $a_1 = 5$, $a_2 = 4$, $b_1 = 0$, $b_2 = 1$, $d_1 = 3$, $d_2 = 8$ y $n = 2$

Una vez obtenidas las trayectorias, tanto las simuladas como las predichas por el modelo, procedemos a superponerlas para una comparación visual directa, tal como se ilustra en la Figura 2.13. Esta superposición es crucial para identificar visualmente las discrepancias entre las trayectorias simuladas y las predicciones del modelo. Además, similar al análisis realizado para los campos vectoriales, se extrajeron las diferencias entre las trayectorias de ambos genes, X y Y. Estas diferencias, que reflejan la variación entre las trayectorias reales y predichas, se presentan detalladamente en la Figura 2.14. Este enfoque analítico nos permite una evaluación minuciosa y detallada de la precisión del modelo en términos de su capacidad para reproducir las dinámicas específicas de cada gen dentro del sistema de Toggle Switch.

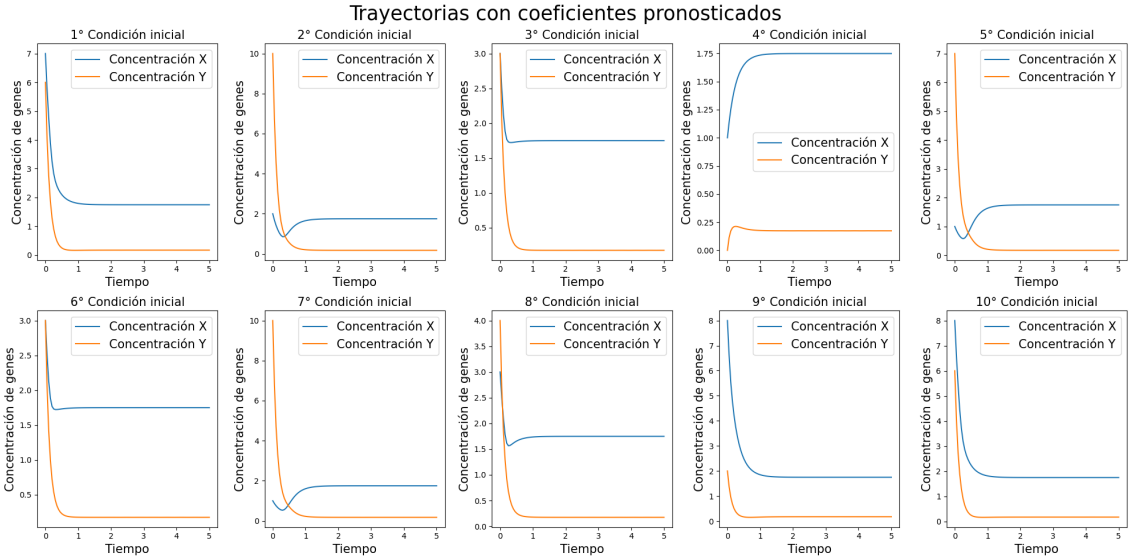


Figura 2.12: Trayectorias de los coeficientes predichos (Modelo convolucional de 100 evaluaciones temporales). Los coeficientes predichos son $a_1 = 6$, $a_2 = 4$, $b_1 = 0$, $b_2 = 1$, $d_1 = 3$, $d_2 = 8$ y $n = 4$.

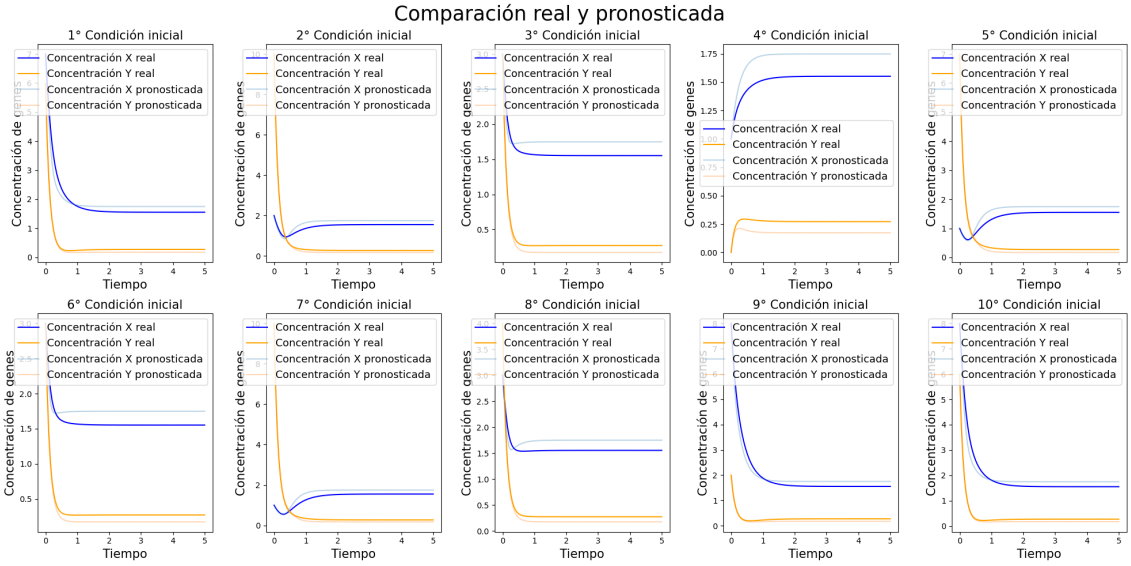


Figura 2.13: Trayectorias reales y predichas por el modelo sobrepuestas. Las reales son las de color más intenso

Por último, la comparación gráfica de los campos vectoriales y trayectorias proporciona una perspectiva visual inmediata de la precisión del modelo. Esta comparativa no solo sirve para validar el modelo, sino que también ofrece insights sobre cómo se comporta el sistema bajo diferentes condiciones. Sin embargo se necesita hacer un análisis estadístico más detallado para poder evaluar de una mejor manera el buen o mal funcionamiento de estas predicciones.

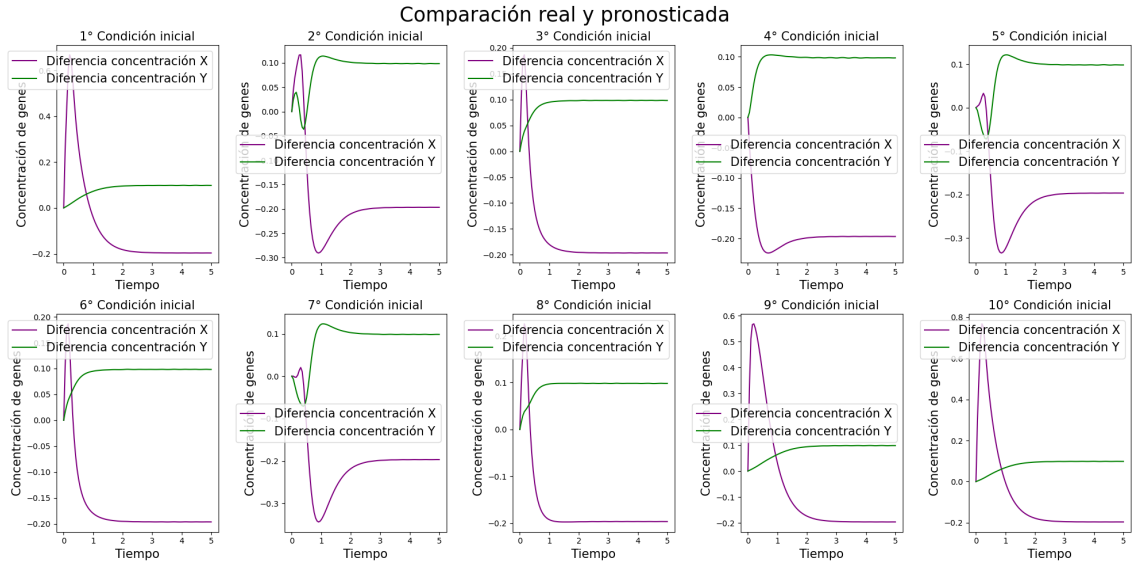


Figura 2.14: Diferencias entre las trayectorias reales y predichas para todas las condiciones iniciales

2.11. Análisis Estadístico Final

Para evaluar exhaustivamente la eficacia de nuestros modelos, se llevó a cabo un análisis estadístico enfocado en la precisión de las predicciones tanto de los campos vectoriales como de las trayectorias. Este análisis se desarrolló a través de enfoques diferenciados para cada tipo de dato, permitiéndonos comprender en detalle cómo el modelo se comporta en distintos escenarios y contextos. A continuación, se detalla el procedimiento y los hallazgos específicos de este análisis estadístico para cada caso, revelando insights valiosos sobre la capacidad predictiva del modelo en diversas situaciones.

2.11.1. Campos vectoriales

En este segmento, se detalla un análisis estadístico implementado para evaluar la capacidad predictiva del modelo de red neuronal en la generación de campos vectoriales derivados de un sistema de ecuaciones diferenciales. El proceso se realiza de la siguiente manera:

1. **Configuración Inicial:** Se establece un bucle que se ejecuta 10,000 veces para generar diferentes conjuntos de datos y evaluar así el modelo en múltiples escenarios.

2. **Generación de Datos Sintéticos:** En cada iteración, se generan coeficientes aleatorios para el sistema de ecuaciones Toggle Switch, representando parámetros biológicos.

3. **Creación del Campo Vectorial Real:** Con estos coeficientes, se calcula un campo vectorial que refleja la dinámica del sistema modelado, resolviendo las ecuaciones diferenciales en un rango específico de valores.

4. **Predicción del Modelo:** Se utiliza el modelo de red neuronal, previamente entrenado, para predecir los coeficientes a partir de los datos del campo vectorial.

5. **Generación del Campo Vectorial Predicho:** Utilizando los coeficientes predichos, se genera un campo vectorial que idealmente debería parecerse al real.

6. **Comparación y Análisis de Precisión:** Se compara el campo vectorial real con el predicho, calculando su diferencia en magnitud y dirección, y se identifican los puntos donde la diferencia relativa es mayor al 5 %.

7. **Resultados y Conclusiones:** Se registran los porcentajes de puntos que exceden este umbral en cada iteración y se calcula un promedio para obtener una medida general de precisión del modelo.

En conclusión, el objetivo de este análisis es determinar qué porcentaje de los datos predichos por el modelo difiere más de un 5 % con respecto a los datos reales. Esta métrica proporciona una estimación cuantitativa de la precisión del modelo en su capacidad de replicar la dinámica compleja inherente a los sistemas de ecuaciones diferenciales que estamos estudiando. Una discrepancia mayor al 5 % indica áreas donde el modelo puede requerir ajustes adicionales o donde los datos pueden ser intrínsecamente más desafiantes de predecir.

2.11.2. Trayectorias

Para las bases de datos de trayectorias, se desarrolló un análisis estadístico con el fin de evaluar la precisión del modelo. Se generaron datos aleatorios para simular condiciones experimentales, y se calculó la diferencia relativa entre los valores de trayectoria reales y los predichos por el modelo. Este proceso se repitió para 1000 iteraciones, con el objetivo de obtener una visión más completa de la eficacia del modelo.

En cada iteración, se definieron condiciones iniciales aleatorias y se calcularon las trayectorias reales a partir de ellas usando el método de integración `solve_ivp`. Posteriormente, se introdujeron estos datos en el modelo entrenado para predecir los coeficientes de las ecuaciones diferenciales. Con los coeficientes predichos, se generaron nuevas trayectorias para compararlas con las reales.

Para evaluar la precisión del modelo en las trayectorias de los genes X y Y, se realizó un cálculo detallado de la diferencia relativa entre los valores reales y los pronosticados. Esta diferencia relativa se calculó para cada punto a lo largo de las trayectorias, tanto para X como para Y, permitiendo así una evaluación exhaustiva del modelo en distintas condiciones.

Se procedió a analizar estos datos para obtener los errores relativos medios y máximos, proporcionando así una medida cuantitativa del desempeño del modelo. Además, se identificó el porcentaje de puntos en los que la diferencia relativa superaba el 100 %. Esta métrica es crucial, ya que resalta los casos donde hay discrepancias significativas entre los datos predichos y los reales, lo que indica potenciales áreas de mejora del modelo.

Un aspecto importante a considerar es la influencia de errores extremadamente grandes en el cálculo del error medio. Por ejemplo, situaciones donde los valores reales son muy pequeños (del orden de centésimas) y los valores predichos son considerablemente más grandes (del orden de decenas) pueden crear una disparidad significativa. Este tipo de discrepancias, aunque poco frecuentes, podrían distorsionar el promedio del error relativo. Por ello, se tomó la decisión de excluir aquellos casos donde la diferencia relativa excediera un umbral específico, en este caso, 100 %. Esto asegura que el cálculo del error relativo medio no se vea desproporcionadamente afectado por unos pocos casos atípicos, proporcionando una evaluación más equilibrada y representativa de la precisión del modelo.

Este análisis estadístico detallado proporciona una evaluación exhaustiva de la capacidad del modelo para reproducir con precisión las dinámicas complejas de los sistemas biológicos modelados. Los resultados de este análisis no solo validan la eficacia del modelo, sino que también identifican áreas de mejora potencial para futuras iteraciones de desarrollo del modelo.

2.12. Creación de base de datos con ruido

Se crearon también bases de datos con ruido, tanto para las trayectorias como para los campos vectoriales. Este enfoque se adoptó con el objetivo de reflejar más fielmente las incertidumbres y las variabilidades inherentes al entorno experimental real. Al incorporar estas perturbaciones, se busca emular con mayor precisión los posibles errores y desviaciones que surgen típicamente en el curso de experimentos prácticos, permitiendo así una aproximación más realista y robusta en la modelización y análisis de datos experimentales.

2.12.1. Trayectorias

Utilizamos una versión modificada del método de Euler, conocida como Euler Maruyama, para integrar estocásticamente las ecuaciones diferenciales del sistema. En cada paso de tiempo (dt), calculamos las nuevas concentraciones de cada gen $(X[i+1], Y[i+1])$ a partir de las concentraciones actuales, los parámetros del sistema y un término de ruido gaussiano. Este término de ruido introduce variaciones en las trayectorias que simulan las fluctuaciones presentes en los sistemas biológicos reales, imitando así la naturaleza impredecible y variable de estos sistemas.

Las trayectorias estocásticas generadas se pueden visualizan en un gráfico, donde se plotean las concentraciones de los genes X e Y en función del tiempo. Esta representación gráfica permite observar la evolución de las concentraciones bajo la influencia del ruido, reflejando la naturaleza impredecible y variable de los sistemas biológicos reales.

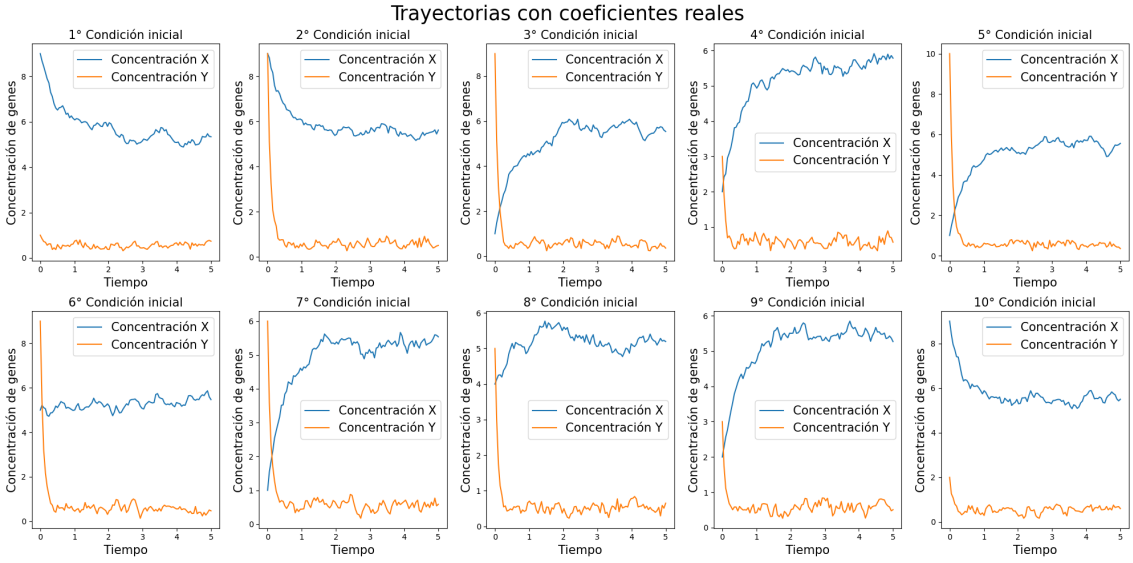


Figura 2.15: Trayectorias estocásticas para las concentraciones de los genes X (azul) y Y (naranja) en un sistema de toggle switch bajo 10 condiciones iniciales distintas, ilustrando la variabilidad debido al ruido estocástico.

En la Figura 2.15 se presentan ejemplos de los datos generados con ruido. Estos datos conservan la estructura básica de la base de datos determinista original, pero se diferencian en la incorporación del ruido para simular variaciones experimentales. Así, se simulan trayectorias para 10 condiciones iniciales distintas, evaluadas en 200, 100 y 50 puntos temporales respectivamente. Esto resulta en conjuntos de datos con dimensiones (10000, 10, 2, 200), (10000, 10, 2, 100) y

(10000, 10, 2, 50). Cada conjunto representa un escenario distinto con un nivel de detalle temporal diferente, lo que permite explorar el impacto del ruido en diversas escalas temporales y condiciones iniciales.

2.12.2. Campos vectoriales

Para la generación de campos vectoriales estocásticos que modelan el comportamiento del sistema de toggle switch, se llevó a cabo un procedimiento iterativo en el que se simularon las dinámicas de las concentraciones de dos genes, X e Y, bajo la influencia de un ruido aleatorio, con el objetivo de emular la variabilidad intrínseca a los experimentos reales.

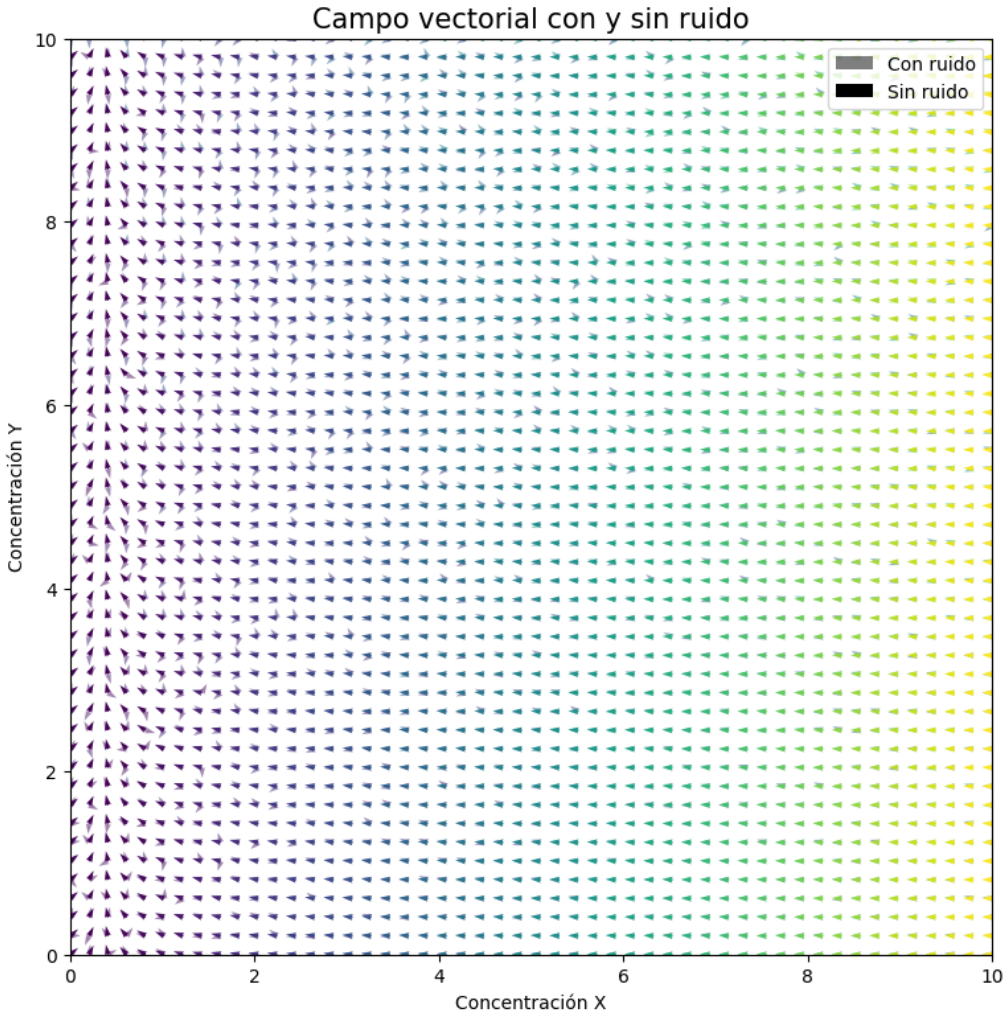


Figura 2.16: Campo vectorial con ruido de un sistema de Toggle Swich, junto con el campo sin ruido.

Durante cada iteración, se seleccionaron aleatoriamente parámetros del modelo $a_1, a_2, b_1, b_2, d_1, d_2$, y n dentro de un rango predefinido. Con estos parámetros, se generó un espacio de fase bidimensional utilizando una malla de puntos cubriendo un rango específico de

concentraciones para X e Y. Para cada punto de esta malla, se calculó el cambio de concentración de los genes X e Y en el tiempo utilizando las ecuaciones diferenciales ordinarias del sistema de toggle switch. Además, se introdujo un ruido gaussiano con media cero y desviación estándar de 0.5 a las derivadas temporales de las concentraciones para simular el ruido experimental.

Con las derivadas obtenidas y el ruido aplicado, se calculó el campo vectorial normalizado en cada punto de la malla, ilustrando así la dirección y la magnitud del cambio de concentración en el espacio de fase. Este campo vectorial se visualizó mediante un gráfico de quiver, donde la dirección y el color de las flechas representan la dirección y la magnitud del cambio, respectivamente. La figura resultante muestra cómo, a pesar de la naturaleza determinista de las ecuaciones subyacentes, la inclusión del ruido introduce una variabilidad que podría compararse con las fluctuaciones observadas en los datos experimentales.

La Figura 2.16 presenta ejemplos de estos campos vectoriales con y sin ruido. Las bases de datos estocásticas se estructuran de manera similar a las deterministas, pero con la incorporación de ruido, lo que permite simular condiciones experimentales reales de manera más efectiva.

Las bases de datos estocásticas se estructuran de manera similar a las deterministas, con conjuntos correspondientes a 100, 50 y 20 puntos de evaluación en el campo vectorial, conformando matrices de dimensiones (10000, 2, 100, 100), (10000, 2, 50, 50) y (10000, 2, 20, 20), respectivamente.

2.13. Creación de redes neuronales datos estocásticos

En el desarrollo de nuestras redes neuronales para el análisis de datos estocásticos, decidimos mantener las mismas arquitecturas que habíamos utilizado exitosamente con los datos deterministas. Esta decisión se basó en la observación de que las configuraciones previas eran suficientemente robustas y adaptables, lo cual nos permitió evitar un nuevo proceso de optimización con Optuna.

El principal reto con los datos estocásticos fue enseñar a las redes neuronales a reconocer y procesar efectivamente las variaciones aleatorias inherentes a estos datos. Esta habilidad es crucial para mejorar la robustez del modelo, especialmente en contextos donde las condiciones experimentales reales presentan niveles significativos de incertidumbre y variabilidad. Al entrenar con datos que incorporan ruido, las redes neuronales aprenden a discernir patrones significativos y tendencias a pesar de la presencia de fluctuaciones aleatorias. Este aprendizaje es fundamental para que el modelo pueda aplicarse con éxito en situaciones reales, donde las condiciones ideales rara vez se cumplen.

La incorporación de datos estocásticos en el entrenamiento de las redes neuronales también sirvió para evaluar y mejorar la capacidad de generalización del modelo. Al enfrentarse a datos con variaciones impredecibles, las redes neuronales deben aprender a adaptarse y responder de manera efectiva, una cualidad que es esencial para aplicaciones prácticas en campos como la biología de sistemas y la genética sintética. Esta capacidad de adaptación y manejo de la incertidumbre no solo incrementa la precisión del modelo en entornos controlados, sino que también asegura su relevancia y aplicabilidad en entornos experimentales más complejos y realistas.

el entrenamiento de redes neuronales con datos estocásticos representa un paso importante hacia el desarrollo de modelos más robustos y adaptables. Este enfoque no solo mejora la precisión del modelo en condiciones ideales, sino que también garantiza su efectividad en situaciones reales, donde la variabilidad y la incertidumbre son factores omnipresentes. Al abordar estos desafíos, preparamos el camino para aplicaciones más avanzadas y realistas en la investigación científica y el desarrollo tecnológico.

2.14. Pruebas gráficas datos estocásticos

El proceso de evaluación visual con datos estocásticos siguió un enfoque meticuloso, paralelo al empleado con los datos deterministas, pero con consideraciones adicionales para las características únicas de los datos estocásticos. Inicialmente, generamos conjuntos de datos que imitaban las fluctuaciones experimentales típicas, introduciendo variabilidad estocástica en nuestras simulaciones. Estos conjuntos de datos proporcionaron una base realista para probar la eficacia del modelo entrenado.

Una vez generados estos datos, los introducimos en el modelo entrenado para obtener una estimación de los coeficientes del sistema de Toggle Switch. Esta etapa fue crucial, ya que nos permitió evaluar la capacidad del modelo para procesar y analizar datos con variaciones impredecibles. Al insertar los coeficientes estimados en el sistema de ecuaciones, se generaron campos vectoriales y trayectorias que, aunque deterministas en su formulación, reflejaban las condiciones estocásticas de los datos de entrada.

La comparación visual entre estas trayectorias teóricas y los datos estocásticos reales fue un paso fundamental en nuestra evaluación. Esta comparación nos permitió observar cómo el modelo manejaba los datos con ruido y fluctuaciones, y en qué medida podía replicar y predecir comportamientos experimentales reales. La habilidad del modelo para alinearse estrechamente con los datos estocásticos proporcionaba una medida valiosa de su precisión y aplicabilidad en situaciones experimentales.

Las trayectorias estocásticas y su comparación con las predicciones del modelo se ilustran en la Figura 2.17. Esta visualización gráfica no solo confirmó la capacidad del modelo para replicar las dinámicas del sistema, sino que también resaltó las áreas donde las predicciones podrían mejorarse. Por otro lado, la correspondencia entre los campos vectoriales estocásticos y aquellos determinados por el modelo, mostrada en la Figura 2.18, proporcionó una visión clara y directa de la eficacia del modelo en representar las dinámicas del sistema bajo variaciones estocásticas.

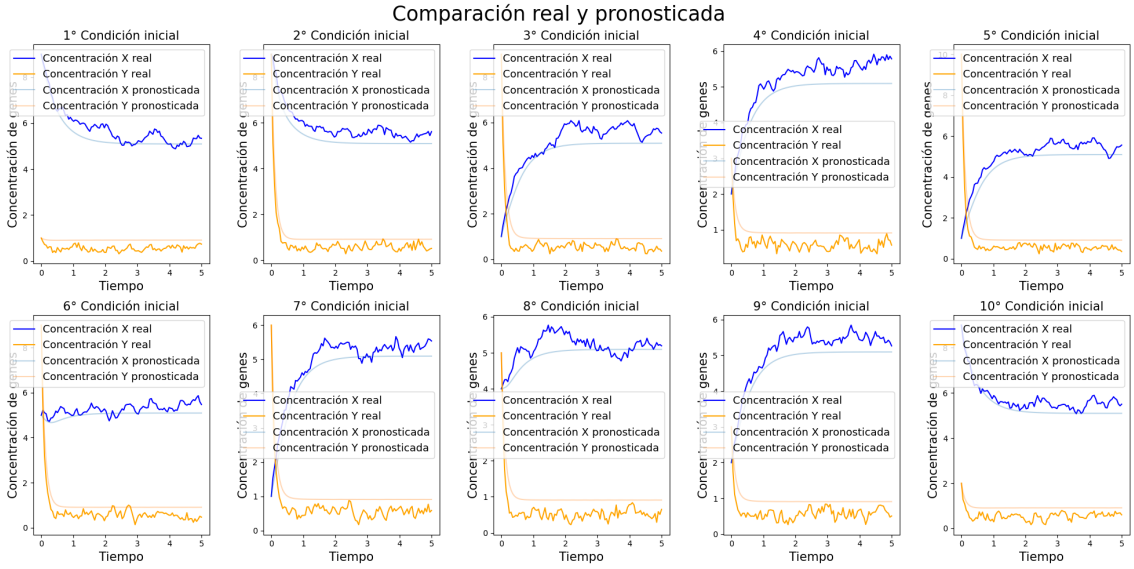


Figura 2.17: Trayectorias estocásticas con sus predicciones deterministas

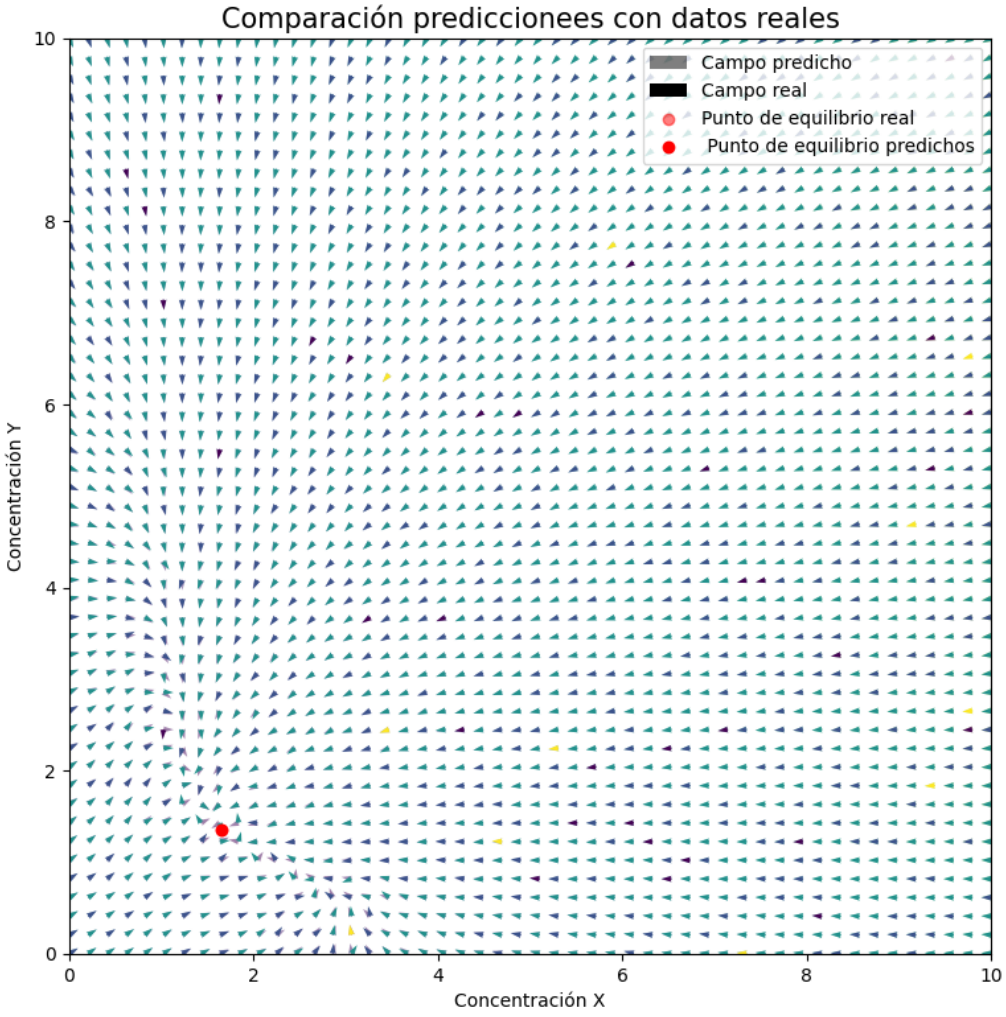


Figura 2.18: Campos vectoriales estocásticas con sus predicciones deterministas.

2.15. Análisis Estadístico Final datos estocásticos

En el análisis estadístico final, adoptamos un procedimiento que se asemeja al enfoque utilizado en el contexto determinista, pero con una consideración adicional hacia la naturaleza estocástica de los datos. Este proceso comienza alimentando el modelo con conjuntos de datos que presentan una variabilidad estocástica. Una vez que el modelo procesa estos datos, genera predicciones que, aunque deterministas en su naturaleza, se basan en la información estocástica suministrada.

Estas trayectorias predichas se comparan luego con las trayectorias deterministas correspondientes a los coeficientes de los datos estocásticos originales, esto para evaluar la capacidad del modelo de capturar la dinámica subyacente a pesar de las fluctuaciones aleatorias.

2.16. Modelo único

En nuestro enfoque, hemos diseñado un modelo de red neuronal que no requiere definir el tamaño exacto de la entrada; es decir, en lugar de necesitar una entrada de dimensiones específicas como (100, 100, 2), nuestro modelo puede manejar entradas de dimensión (None, None, 2), donde 'None' representa cualquier número entero. Esta flexibilidad se logró mediante la implementación de una red neuronal convolucional adaptada, la cual fue optimizada con el uso de Optuna.

El Global Average Pooling (GAP) juega un papel crucial en esta arquitectura. A diferencia del Max Pooling, que extrae el valor máximo de una región específica en la imagen, el GAP calcula el promedio de cada mapa de características, generando un único valor promedio para cada uno. Este enfoque reduce de manera significativa la dimensionalidad de la salida, transformando un conjunto extenso de mapas de características en un vector conciso de características, como se ilustra en la Figura 2.19.

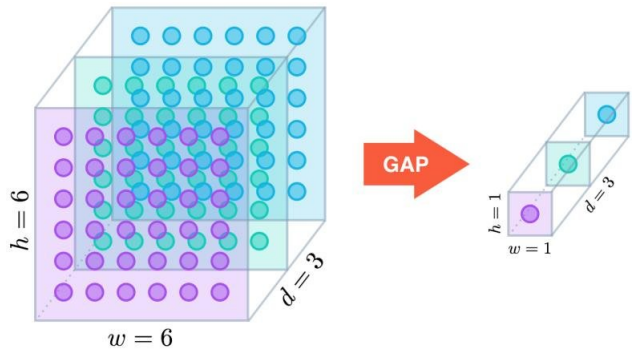


Figura 2.19: Forma en que el Global Average Pooling toma un conjunto de mapas de características en un vector de características.

La adopción del Global Average Pooling en nuestro modelo tiene como objetivo obtener una dimensión final única, independientemente del tamaño de la entrada. Este modelo se aplicó específicamente a los datos de los campos vectoriales, y la elección de enfocarse en este aspecto se discutirá más adelante en la sección de resultados. Cabe destacar que el entrenamiento se realizó con la base de datos de mayor tamaño disponible (100,100,2).

La estructura del modelo es la siguiente:

1. **Primera Capa Convolucional**: Tiene 18 filtros de tamaño (5, 3), activación 'sigmoid', y regularización L2. La entrada acepta datos de tamaño variable con 2 canales. La salida es (None, None, 18).

2. **Primera Capa de Max Pooling**: Con un pool size de (2, 2). La salida sigue siendo (None, None, 18).

3. **Segunda Capa Convolucional**: Con 47 filtros de tamaño (3, 4) y activación 'tanh'. Salida de tamaño (None, None, 47).

4. **Tercera Capa Convolucional**: Utiliza 38 filtros de tamaño (5, 4), también con activación 'tanh'. Salida de tamaño (None, None, 38).

5. **Cuarta Capa Convolucional**: Tiene 50 filtros de tamaño (4, 5) y activación 'relu'. Salida de tamaño (None, None; 50).

6. **Segunda Capa de Max Pooling**: Similar a la primera capa de Max Pooling, la salida es de las mismas dimensiones
7. **Quinta Capa Convolucional**: Con 22 filtros de tamaño (2, 5) y activación 'relu', la salida es de tamaño (None,None, 22).
8. **Tercera Capa de Max Pooling**: Reduce aún más las dimensiones espaciales, enfocándose en las características más significativas.
9. **Primera Capa Densa**: Con 79 neuronas y activación 'tanh'. Con salidas de tamaño (None, None, 79)
10. **Dropout**: Con una tasa de 0.2178, ayuda a prevenir el sobreajuste al desactivar aleatoriamente algunas neuronas durante el entrenamiento.
11. **Segunda Capa Densa**: Similar a la primera, pero con 89 neuronas y una tasa de Dropout de 0.2868. Continúa el procesamiento de los datos para la predicción final. Salida de tamaño (None, None, 89).
12. **Capa de Salida**: Tiene 7 neuronas, una por cada parámetro a predecir. Salida de tamaño (None,None, 7).
13. **Global Average Pooling**: Reduce la salida de la última capa convolucional a un conjunto de valores promedio, uno por canal, para prepararlos para la clasificación final. Salida de tamaño (7), que es el deseado.

La metodología descrita concluye en este punto, preparando el terreno para el siguiente capítulo dedicado a la presentación y análisis de resultados. En esa sección, se examinarán detalladamente los hallazgos obtenidos a través de las diversas técnicas y modelos implementados, proporcionando una visión integral del rendimiento y las implicaciones de la investigación.

2.17. Límites de tamaños de la metodología

Tras confirmar la eficacia de nuestra metodología, surgió la pregunta: ¿Cuál es el tamaño mínimo de experimento requerido para un funcionamiento óptimo de nuestra metodología? Para responder a esta interrogante, generamos bases de datos de menor tamaño, específicamente con 10 y 5 puntos para el campo vectorial y 20 y 10 evaluaciones temporales para las trayectorias, en ambos casos considerando versiones con y sin ruido.

En el contexto del campo vectorial, la estructura de las bases de datos se ilustra en la Figura 2.20 con la base de datos de 10 evaluaciones por componente del lado izquierdo y la base de datos con 5 evaluaciones del lado derecho, tomando los mismos coeficientes para ambos casos, es decir, es el mismo campo pero con distinta resolución.

Por otro lado, la Figura 2.21 muestra una solución del sistema de ecuaciones diferenciales bajo 10 condiciones iniciales distintas con 10 evaluaciones temporales. Se evidencia que las líneas, que antes eran continuas, ahora presentan discontinuidades atribuibles a la reducción en el número de evaluaciones.

En nuestro enfoque para las bases de datos estocásticas, mantenemos la misma metodología para los campos vectoriales como en las bases de datos más grandes. Sin embargo, el tratamiento de las trayectorias en el entorno estocástico difiere significativamente, especialmente en lo que respecta a la selección de los pasos temporales.

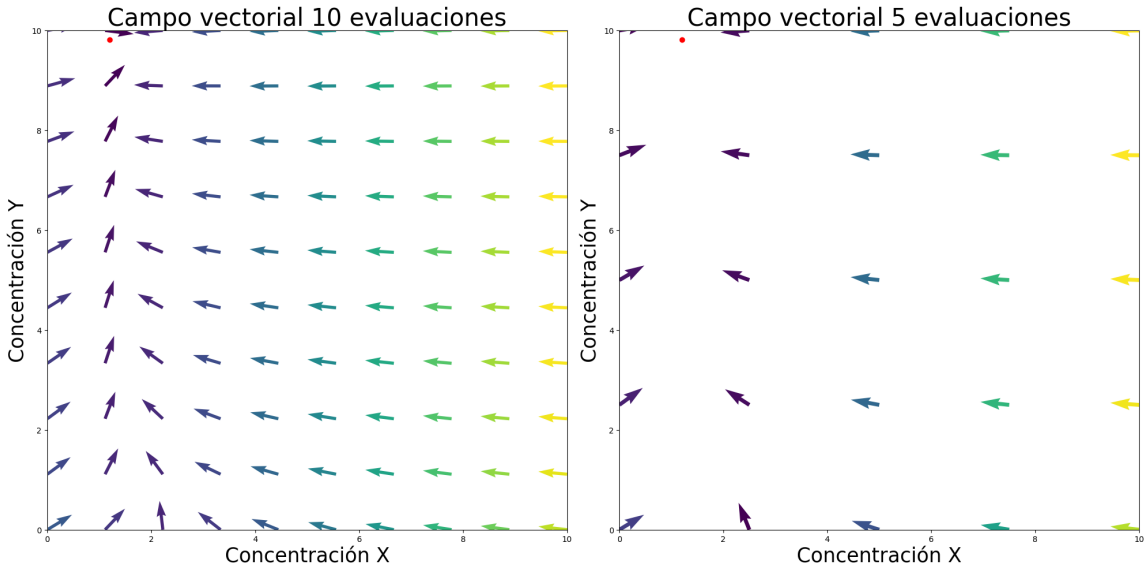


Figura 2.20: Bases de datos de campos vectoriales par (10,10) (izquierda) y (5,5) (derecha) evaluaciones.

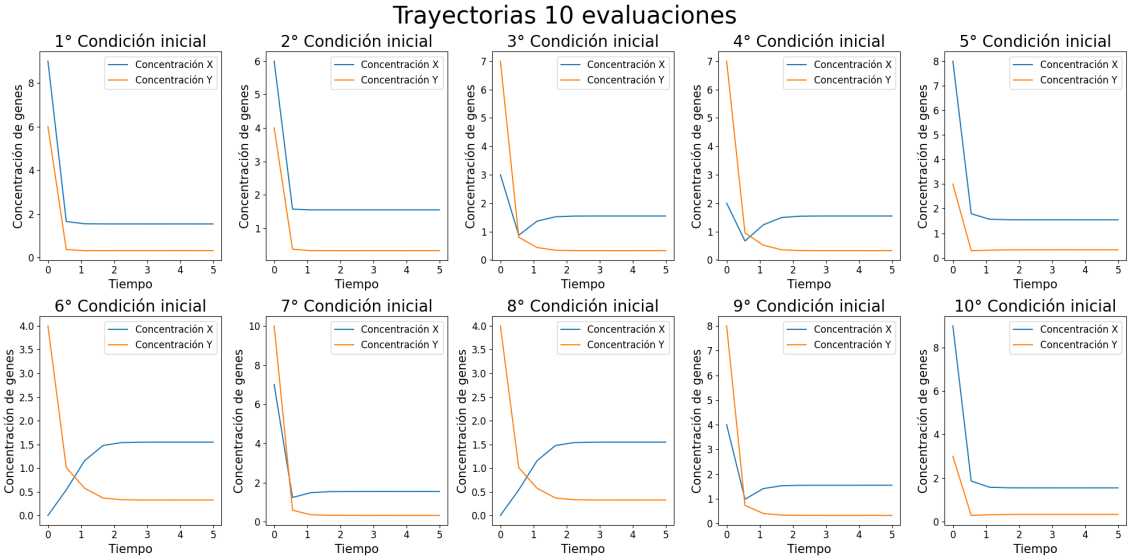


Figura 2.21: Base de datos de trayectorias para 10 evaluaciones temporales deterministas, se pueden observar que las curvas ya no son tan suaves como solían ser para bases de datos más grandes.

Para la simulación de trayectorias simulamos bases de datos más grandes computacionalmente y luego fuimos adquiriendo algunos puntos de estas bases más grandes. Por ejemplo, para el caso de 10 evaluaciones temporales, inicialmente generamos un conjunto de trayectorias con una resolución de 50 puntos ($N = 50$), basándonos en un tiempo total de simulación $T = 5$ y un paso temporal $dt = 0{,}1$. No obstante, para adecuarnos a las restricciones de tamaño de nuestras bases de datos, solo seleccionamos cada quinto punto de estas trayectorias, reduciendo así cada una a 10 puntos.

La elección de un paso temporal más pequeño ($dt = 0{,}1$) en la implementación es crucial para evitar la divergencia del algoritmo de Euler-Mayurama. Cuando se utilizan pasos temporales grandes en este algoritmo, las aproximaciones lineales empleadas no pueden capturar con precisión la complejidad y la naturaleza estocástica de las dinámicas del sistema. En nuestro caso, esto podría conducir a soluciones numéricas que se desvíen significativamente de las trayectorias reales, especialmente dado el comportamiento no lineal y estocástico inherente a nuestro modelo de regulación genética.

Por lo tanto, la decisión de emplear un paso temporal más pequeño y luego seleccionar solo ciertos puntos de la trayectoria es un compromiso entre mantener la precisión y estabilidad del algoritmo y cumplir con las restricciones de tamaño de nuestra base de datos. Este enfoque nos permite generar trayectorias fiables y representativas, a pesar de la reducción en la cantidad de puntos de datos. Un ejemplo de esta base de datos se puede ver en la Figura 2.22 para 10 evaluaciones temporales.

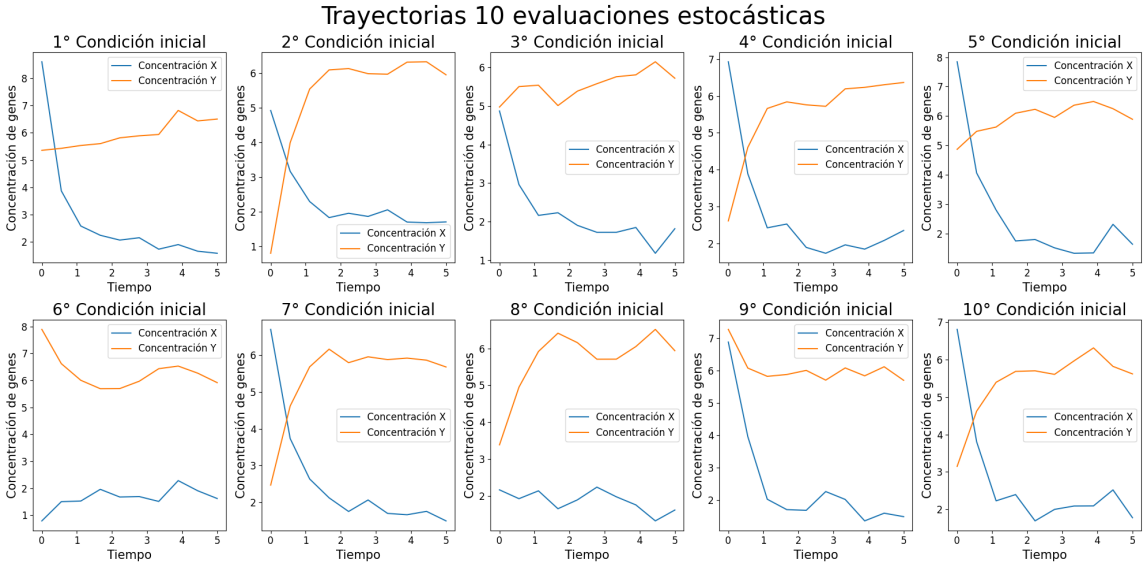


Figura 2.22: Base de datos de trayectorias para 10 evaluaciones temporales, se pueden observar que el ruido se puede observar de manera más significativa entre más pequeña sea la base de datos.

Además de los ajustes mencionados anteriormente, el resto de la metodología se aplicó de manera consistente con los procedimientos establecidos para bases de datos más grandes. El foco principal de esta fase del estudio fue evaluar el impacto de utilizar bases de datos reducidas en la precisión de los coeficientes obtenidos. Esta evaluación es crucial, ya que proporciona insights sobre la robustez de nuestra metodología cuando se enfrenta a limitaciones en la cantidad de datos disponibles.

Nuestro análisis se centró en determinar la magnitud de las desviaciones o errores en los coeficientes estimados, comparándolos con aquellos obtenidos a partir de bases de datos más amplias. Este enfoque nos permitió cuantificar la eficacia de nuestra metodología en condiciones de datos restringidos y, a su vez, identificar posibles áreas de mejora. La fiabilidad de los coeficientes en condiciones de limitación de datos es un aspecto fundamental para validar la aplicabilidad y flexibilidad de nuestro enfoque en diversos contextos experimentales, especialmente en situaciones donde la recopilación de grandes volúmenes de datos puede no ser viable.

Esta parte de la investigación es de especial importancia, ya que proporciona una comprensión

más profunda de cómo la escasez de datos afecta la precisión de los modelos en la práctica, y cómo nuestra metodología se adapta y responde a estas limitaciones.

Capítulo 3

Resultados

En esta sección, se realizará un análisis exhaustivo y comparativo de las métricas finales obtenidas de los datos de prueba, así como de las estadísticas finales detalladas en la sección previa. Este enfoque permitirá evaluar la efectividad de la metodología empleada tanto con datos deterministas como estocásticos con y sin Optuna. Además, se explorará cómo las variaciones en el tamaño del experimento influyen en la precisión y fiabilidad de las predicciones, proporcionando una comprensión más profunda del comportamiento y desempeño del modelo en diferentes escenarios.

3.1. Comparación de Métricas

3.1.1. Metodología determinista

Trayectorias

Sin Optuna:

Modelo	a_1	a_2	b_1	b_2	d_1	d_2	n	**Mean Metric**
Convolucional 200	2.056	2.052	0.889	0.949	0.312	0.278	1.254	0.262
Densa 200	2.298	2.377	1.233	1.248	0.348	0.367	1.376	0.309
Convolucional 100	2.017	2.002	0.896	0.833	0.397	0.295	1.376	0.260
Densa 100	2.285	2.315	1.043	1.061	0.338	0.301	1.301	0.287
Convolucional 50	2.270	2.144	0.961	0.902	0.316	0.291	1.111	0.269
Densa 50	2.457	2.401	1.238	1.131	0.344	0.320	1.374	0.310

Tabla 3.1: Resultados de las métricas para modelos de trayectorias deterministas sin Optuna.

Con Optuna:

Modelo	a_1	a_2	b_1	b_2	d_1	d_2	n	**Mean Metric**
Convolucional 200	1.956	1.893	0.812	0.860	0.331	0.357	1.157	0.230
Densa 200	2.086	2.114	1.180	1.056	0.450	0.409	1.155	0.264
Convolucional 100	1.806	1.806	0.809	0.755	0.385	0.387	1.145	0.219
Densa 100	2.057	2.085	1.070	0.995	0.404	0.453	1.164	0.253
Convolucional 50	1.724	1.855	0.753	0.754	0.391	0.338	1.145	0.221
Densas 50	2.088	2.283	1.002	1.032	0.325	0.357	1.144	0.260

Tabla 3.2: Resultados de las métricas para modelos de trayectorias deterministas con Optuna.

El análisis revela que la aplicación de Optuna en la metodología para trayectorias deterministas conduce a una ligera mejora en las métricas. Sin embargo, se observa que el aumento en el volumen de datos no implica necesariamente mejoras significativas en los resultados.

Otro aspecto relevante que surge del análisis es la superioridad de las redes convolucionales sobre las redes densas en términos de rendimiento. Los resultados indican que las redes convolucionales logran mejores métricas en comparación con las redes densas, destacando su eficacia en este contexto específico de estudio.

Campos vectoriales

Sin Optuna:

Modelo	a_1	a_2	b_1	b_2	d_1	d_2	n	**Mean Metric**
Función de Costo 100	3.687	3.548	0.975	1.956	0.099	0.118	2.452	0.424
Convolucional 100	0.577	0.550	0.389	0.369	0.209	0.183	0.210	0.082
Densa 100	0.642	0.533	0.361	0.369	0.130	0.211	0.651	0.096
Función de Costo 50	3.032	2.855	0.856	1.019	0.089	0.100	1.757	0.320
Convolucional 50	1.384	1.316	1.054	1.088	0.735	0.762	1.339	0.253
Densa 50	0.888	0.839	0.684	0.701	0.561	0.333	0.885	0.161
Función de Costo 20	2.780	2.761	2.775	2.790	1.000	1.079	1.842	0.493
Convolucional 20	0.618	0.644	0.408	0.402	0.222	0.213	0.331	0.093
Densa 20	0.802	0.748	0.565	0.469	0.369	0.382	1.153	0.147

Tabla 3.3: Resultados de las métricas para modelos de campos vectoriales deterministas sin Optuna.

Con Optuna:

Modelo	a_1	a_2	b_1	b_2	d_1	d_2	n	**Mean Metric**
Función de Costo 100	2.472	2.391	0.348	0.323	0.050	0.169	1.211	0.217
Convolucional 100	0.050	0.037	0.062	0.058	0.047	0.037	0.024	0.010
Densas 100	0.162	0.164	0.161	0.169	0.146	0.143	0.120	0.033
Función de Costo 50	2.516	2.475	0.385	0.418	0.050	0.061	1.271	0.223
Convolucional 50	0.045	0.036	0.049	0.050	0.032	0.027	0.023	0.008
Densa 50	0.196	0.163	0.165	0.183	0.183	0.202	0.127	0.038
Función de Costo 20	2.416	2.339	1.177	1.052	0.135	0.123	1.800	0.296
Convolucional 20	0.539	0.538	0.305	0.332	0.126	0.116	0.132	0.069
Densa 20	0.579	0.581	0.324	0.322	0.091	0.104	0.177	0.072

Tabla 3.4: Resultados de las métricas para modelos de campos vectoriales determinista con Optuna.

Por otro lado, en el caso de los campos vectoriales, las mejoras obtenidas mediante la optimización con Optuna son significativas. Además, se evidencia que las redes neuronales, en términos generales, tienden a producir resultados más favorables al trabajar con campos vectoriales en comparación con las trayectorias.

Es notable mencionar que la función de costo personalizada, tanto en escenarios con como sin Optuna, presenta los resultados menos óptimos en nuestras pruebas. Esta es una de las razones por la cual no fue desarrollada una función de costo personalizada para el caso de las trayectorias.

3.1.2. Metodología estocástica

En esta etapa, optamos por omitir los resultados obtenidos sin la optimización proporcionada por Optuna. La decisión se basa en las observaciones realizadas en la sección dedicada a los datos deterministas, donde se evidenció una mejora sustancial en el desempeño con los modelos optimizados por Optuna. Por lo tanto, consideramos que incluir los resultados sin Optuna no aportaría valor significativo al análisis, enfocándonos en resaltar las mejoras y diferencias que Optuna aporta al rendimiento de las redes.

Trayectorias

Modelo	a_1	a_2	b_1	b_2	d_1	d_2	n	**Mean Metric**
Convolucional 200	2.095	2.090	0.952	0.949	0.384	0.391	1.157	0.250
Densa 200	2.067	2.100	1.159	1.035	0.414	0.423	1.156	0.261
Convolucional 100	2.270	2.286	1.086	1.026	0.380	0.391	1.321	0.289
Densa 100	2.317	2.409	1.232	1.151	0.378	0.390	1.364	0.306
Convolucional 50	2.422	2.330	1.151	1.197	0.442	0.395	1.384	0.307
Densa 50	2.407	2.400	1.233	1.279	0.420	0.443	1.399	0.316

Tabla 3.5: Resultados de las métricas para modelos de trayectorias estocásticos con Optuna.

Campos vectoriales

Modelo	a_1	a_2	b_1	b_2	d_1	d_2	n	**Mean Metric**
Función de Costo 100	1.899	1.889	1.210	1.161	0.723	0.697	1.463	0.279
Convolucional 100	0.479	0.485	0.423	0.401	0.328	0.320	0.360	0.086
Densas 100	0.838	0.841	0.739	0.741	0.660	0.654	0.567	0.156
Función de Costo 50	1.641	1.596	1.043	1.339	0.686	0.677	1.332	0.261
Convolucional 50	0.309	0.312	0.209	0.211	0.134	0.135	0.279	0.050
Densa 50	0.831	0.881	0.749	0.749	0.687	0.656	0.566	0.161
Función de Costo 20	1.339	1.443	0.881	0.866	0.609	0.599	1.706	0.232
Convolucional 20	0.558	0.551	0.441	0.404	0.314	0.331	0.521	0.097
Densas 20	0.796	0.818	0.692	0.717	0.635	0.645	0.526	0.151

Tabla 3.6: Resultados de las métricas para modelos de campos vectoriales estocásticos con Optuna.

De nuevo podemos ver que los resultados indican claramente que los modelos aplicados a los campos vectoriales son más eficaces para predecir los coeficientes del sistema toggle switch, al igual que las redes convolucionales ofrecen mejores resultados.

Es notorio en ambos casos que los modelos enfrentan mayores desafíos al predecir con precisión los coeficientes a_1, a_2, b_1, y b_2 del sistema toggle switch [2.1]. Esta tendencia sugiere que estos parámetros específicos representan un nivel de complejidad más elevado para el aprendizaje y la predicción de los modelos en cuestión.

Para poder visualizar mejor las diferencias entre los datos estocásticos con los deterministas, podemos observar en la Tabla 3.7 la métrica Mean Metric para ambos para el caso de las trayectorias, y el la Tabla 3.8 la métrica para el caso de los campos vectoriales.

Modelo	Mean Metric Determinista	Mean Metric Estocástico
Convolucional 200	0.230	0.250
Densas 200	0.264	0.261
Convolucional 100	0.219	0.289
Densa 100	0.253	0.306
Convolucional 50	0.221	0.307
Densas 50	0.260	0.316

Tabla 3.7: Comparación de la métrica 'Mean Metric' para las trayectorias deterministas y estocásticas.

Modelo	Mean Metric Determinista	Mean Metric Estocástico
Función de Costo 100	0.217	0.279
Convolucional 100	0.010	0.086
Densas 100	0.033	0.156
Función de Costo 50	0.223	0.261
Convolucional 50	0.008	0.050
Densa 50	0.038	0.161
Función de Costo 20	0.296	0.232
Convolucional 20	0.069	0.097
Densas 20	0.072	0.151

Tabla 3.8: Comparación de la métrica 'Mean Metric' para los campos vectoriales deterministas y estocásticos.

Los resultados muestran que, aunque hay diferencias en los valores obtenidos, estas no son extremas. Esto indica que nuestro modelo es efectivo al interpretar datos con ruido, demostrando su capacidad para manejar variaciones en los datos experimentales de manera eficiente.

3.2. Análisis Estadístico Final

En esta sección, se presenta el análisis estadístico final, tal como se detalla en la sección de Metodología. Este análisis exhaustivo tiene como objetivo evaluar la confiabilidad y precisión de nuestras predicciones, proporcionando una comprensión más profunda de la efectividad de los modelos desarrollados.

3.2.1. Metodología determinista

Trayectorias

Para facilitar la comprensión de las tablas asociadas con el análisis de trayectorias, se utilizan los siguientes términos:

- **Error X:** Representa el error relativo medio en la variable X.
- **Error Y:** Denota el error relativo medio en la variable Y.
- **Dif.** $\geq 100\,\%$ **X:** Hace referencia al porcentaje de puntos cuya diferencia relativa es mayor o igual al $100\,\%$ respecto a todas las condiciones iniciales en X.
- **Dif.** $\geq 100\,\%$ **Y:** Alude al porcentaje de puntos con una diferencia relativa mayor o igual al $100\,\%$ en la variable Y.

Modelo	**Error X**	**Error Y**	**Dif.** $\geq 100\,\%$ **X**	**Dif.** $\geq 100\,\%$ **Y**
Convolucional 200	12.62 %	14.94 %	1.57 %	3.42 %
Densas 200	15.23 %	15.01 %	4.41 %	4.00 %
Convolucional 100	15.45 %	17.93 %	3.27 %	3.36 %
Densas 100	16.88 %	13.90 %	2.92 %	3.83 %
Convolucional 50	13.48 %	14.30 %	2.74 %	2.66 %
Densa 50	14.61 %	14.75 %	2.96 %	3.23 %

Tabla 3.9: Análisis estadístico de los errores relativos de trayectorias. Error X, Y representa el error relativo medio de X, Y, mientras que Dif. $\geq 100\,\%$ X,Y el porcentaje de puntos cuyo error relativo es mayor al 100 %

La observación de los resultados indica que el modelo presenta una mayor dificultad al predecir las trayectorias asociadas al gen Y en comparación con el gen X. Además, en promedio, se encuentra que existe una discrepancia del 15 % entre las trayectorias experimentales y las predicciones realizadas por el modelo. Esta diferencia subraya ciertos desafíos en la precisión del modelo al abordar la dinámica del gen Y.

En general, el modelo muestra una capacidad efectiva para manejar la complejidad inherente a la regulación genética, aunque con margen para optimizaciones y mejoras en futuras investigaciones.

Campos vectoriales

Modelo	**Promedio de Porcentajes**
Función de Costo 100	90.48 %
Convolucional 100	4.72 %
Densas 100	5.18 %
Función de Costo 50	91.12 %
Convolucional 50	4.58 %
Densa 50	9.25 %
Función de Costo 20	24.47 %
Convolucional 20	2.87 %
Densas 20	3.12 %

Tabla 3.10: Análisis estadístico de los errores relativos de campos vectoriales.

Es importante recordar que la fila 'Promedio de Porcentajes' en la tabla se refiere al promedio de casos en los que la diferencia entre los campos vectoriales predichos y los reales supera el 5 %. Este indicador es crucial para evaluar la precisión de las predicciones del modelo en términos de cómo se comparan con los datos reales y muestra la proporción de predicciones que presentan desviaciones significativas.

Nuevamente, se observa que la aplicación de la función de costo personalizada no contribuye a mejorar las predicciones, sino que tiende a disminuir su precisión. Sin embargo, a pesar de esto, los porcentajes de error resultan ser relativamente bajos, lo que sugiere que, en general, las predicciones realizadas por el modelo son bastante acertadas.

3.2.2. Metodología estocástica

Trayectorias

Modelo	Error X	Error Y	Dif. $\geq$ 100 % X	Dif. $\geq$ 100 % Y
Convolucional 200	14.18 %	14.46 %	2.87 %	2.00 %
Densas 200	15.52 %	14.42 %	4.87 %	4.63 %
Convolucional 100	14.70 %	15.67 %	3.52 %	1.68 %
Densas 100	15.70 %	14.94 %	5.14 %	6.00 %
Convolucional 50	13.72 %	14.79 %	1.98 %	2.54 %
Densas 50	16.45 %	15.79 %	5.83 %	4.33 %

Tabla 3.11: Análisis estadístico de los modelos para trayectorias estocásticas

Aquí podemos ver como si hay una pequeña pérdida de presición con respecto a la metodología determinista, sin embargo aún siguen siendo resultados acepables con estos datos estocáticos.

Campos vectoriales

Modelo	Promedio de Porcentajes (%)
Función de Costo 100	42.39 %
Convolucionales 100	11.40 %
Densas 100	13.39 %
Función de Costo 50	41.00 %
Convolucional 50	8.23 %
Densa 50	11.95 %
Función de Costo 20	30.33 %
Convolucional 20	13.70 %
Densa 20	15.87 %

Tabla 3.12: Análisis estadístico de los modelos para campos vectoriales estocásticos

En esta situación, aunque el desempeño predictivo con los datos deterministas es ligeramente inferior comparado con los estocásticos, los resultados siguen siendo positivos. Es importante destacar que los porcentajes presentados reflejan la proporción de puntos cuya diferencia relativa supera el 5 % entre las predicciones y los datos reales. A pesar de esta pequeña disminución en la precisión, los modelos mantienen un nivel de rendimiento satisfactorio, lo que indica una buena capacidad de adaptación y aprendizaje incluso en condiciones menos controladas y predecibles.

3.3. Modelo único

Lamentablemente, los resultados obtenidos con el modelo único no cumplieron con las expectativas iniciales, lo que motivó nuestra decisión de limitar su aplicación exclusivamente a los campos vectoriales y con los datos deterministas. A pesar de este contratiempo, se llevó a cabo un análisis detallado utilizando este modelo, pero enfocándonos en distintos tamaños de datos para comprender mejor su comportamiento y rendimiento.

Para profundizar en esta investigación, se realizaron las mismas pruebas estadísticas descritas anteriormente para los datos deterministas variando las dimensiones de los datos desde 40 hasta 100, incrementando en intervalos de 5. Esta metodología nos permitió evaluar la capacidad del

modelo para manejar diferentes escalas de datos y su impacto en la precisión de las predicciones. Los resultados de estas pruebas proporcionan una visión valiosa sobre la escalabilidad del modelo y su efectividad en diferentes contextos de tamaño de datos.

Los resultados específicos de estas pruebas, que incluyen el análisis estadístico de los campos vectoriales para cada tamaño de datos evaluado, se presentan a continuación, proporcionando una panorámica integral de la eficacia del modelo en diversas condiciones.

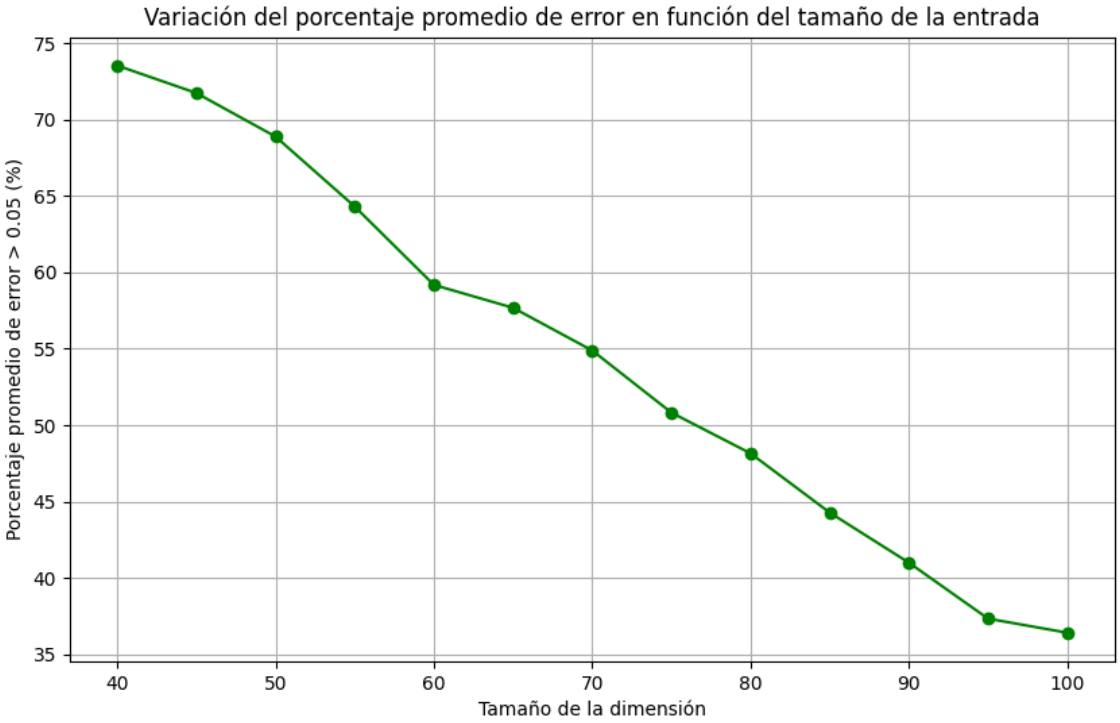


Figura 3.1: Error Promedio Relativo por tamaño de datos para el modelo único. Se puede observar una disminución del Error Promedio Relativo con el Aumento del Tamaño de Entrada de la Base de Datos, sin embargo siguen estando muy lejos de los valores alcanzados con redes independientes

Este análisis de los resultados nos lleva al siguiente capítulo de la tesis, donde consolidaremos nuestras observaciones y reflexionaremos sobre las implicaciones y conclusiones derivadas de este estudio.

3.4. Límite de tamaños de la metodología

En esta sección, presentamos una comparativa detallada entre los resultados obtenidos de las bases de datos descritas en la sección anterior y aquellos derivados de las bases de datos de menor tamaño. La Tabla 3.13 muestra un análisis comparativo de la métrica 'Mean Metric' para las trayectorias, tanto en las bases de datos deterministas como en las estocásticas. Esta métrica proporciona una visión cuantitativa sobre la precisión y el comportamiento de nuestras simulaciones bajo diferentes condiciones de tamaño de datos.

Adicionalmente, la Figura 3.2 ilustra gráficamente estas comparaciones, facilitando una interpretación visual más intuitiva. Esta representación gráfica es especialmente útil para comprender

cómo las variaciones en el tamaño de la base de datos influyen en las métricas evaluadas, ofreciendo una perspectiva más clara de las tendencias y patrones que emergen bajo diferentes condiciones de escala de datos.

Modelo	Mean Metric Determinista	Mean Metric Estocástico
Convolucional 200	0.230	0.250
Convolucional 100	0.219	0.289
Convolucional 50	0.221	0.307
Convolucional 20	0.224	0.2192
Convolucional 10	0.2393	0.2237

Tabla 3.13: Comparación de la métrica 'Mean Metric' para las trayectorias deterministas y estocásticas

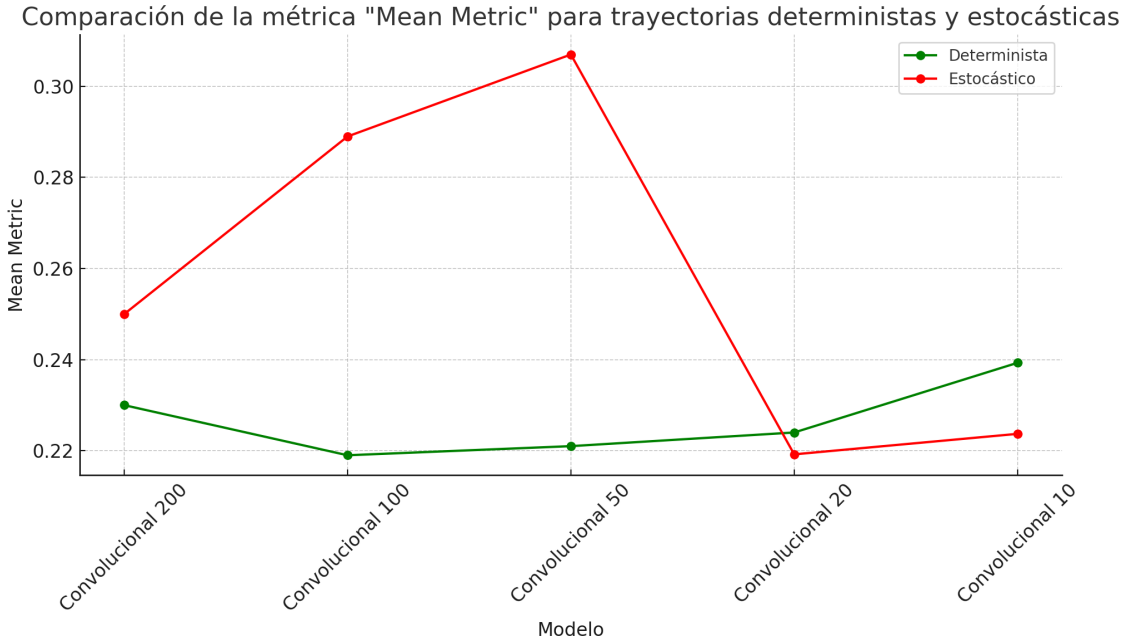


Figura 3.2: Comparación entre las métricas 'Mean Metric' obtenidas de trayectorias deterministas y estocásticas en modelos convolucionales de diferentes tamaños. Los modelos varían desde 200 hasta 10 unidades, representados en el eje horizontal. La línea verde representa los valores obtenidos bajo el enfoque determinista, mientras que la línea roja muestra los valores para el enfoque estocástico.

La Figura anterior ilustra una tendencia interesante: a medida que disminuye el tamaño de la base de datos determinista, se observa un incremento en la métrica 'Mean Metric'. Sin embargo, este aumento es relativamente modesto, sugiriendo que nuestra metodología mantiene un buen rendimiento incluso con bases de datos de menor tamaño en el contexto de las trayectorias deterministas. Este hallazgo respalda la eficacia de nuestro enfoque bajo condiciones de limitación de datos.

En cuanto a las bases de datos estocásticas, se percibe un patrón similar, aunque con una variación notable: para las bases de datos más recientes (con 20 y 10 evaluaciones temporales), se registra una disminución en la métrica. Este fenómeno puede atribuirse al uso de Optuna en estas bases de datos, en contraste con la estructura de red empleada para las bases de datos más grandes (200, 100 y 50 evaluaciones temporales), ya que se usaron las mismas redes que para las

bases de datos deterministas. Este resultado destaca la eficacia de Optuna en la optimización y diseño de redes neuronales, demostrando su capacidad para adaptarse y mejorar el rendimiento en escenarios con restricciones de datos más severas.

La observación de estos patrones nos permite no solo entender mejor la robustez de nuestra metodología en diferentes escenarios, sino también reconocer el potencial de herramientas avanzadas como Optuna en la mejora y adaptación de redes neuronales para la investigación en regulación genética.

Para el caso de los campos vectoriales, la comparación de la métrica Mean Metric se puede observar en la Tabla 3.14, mientras que su respectiva gráfica en la Figura 3.3.

Modelo	Mean Metric Determinista	Mean Metric Estocástico
Convolucional 100	0.010	0.086
Convolucional 50	0.008	0.050
Convolucional 20	0.069	0.097
Convolucional 10	0.075	0.152
Convolucional 5	0.022	0.1947

Tabla 3.14: Comparación de la métrica 'Mean Metric' para los campos vectoriales deterministas y estocásticos

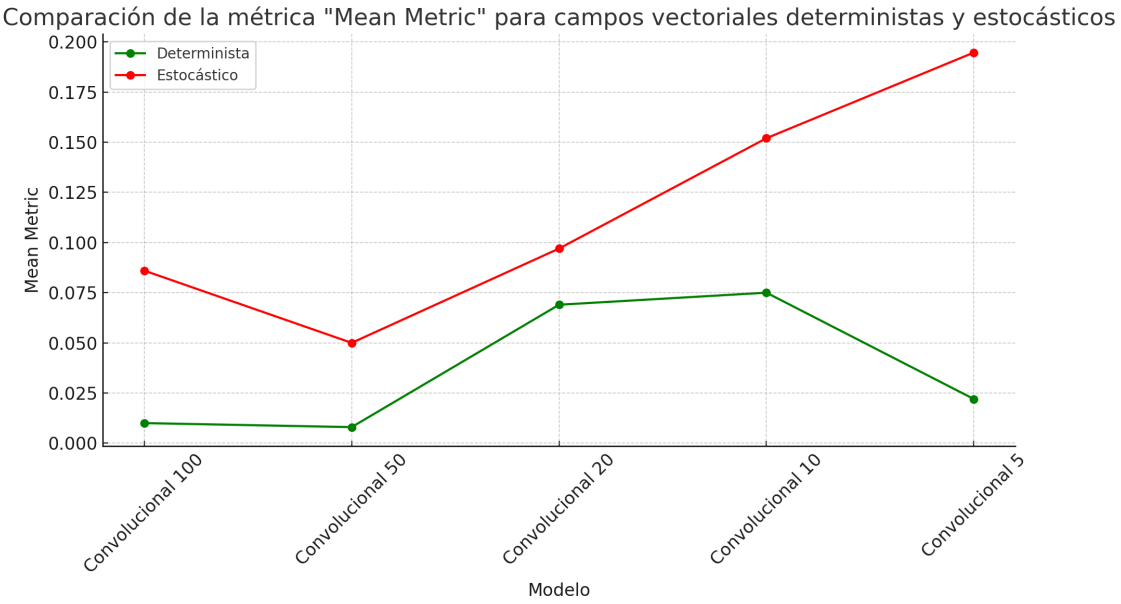


Figura 3.3: Comparativa de la métrica 'Mean Metric' entre enfoques deterministas y estocásticos en modelos convolucionales de diferentes tamaños para campos vectoriales, que van desde 100 hasta 5 unidades. La línea verde muestra los resultados para el enfoque determinista y la línea roja para el estocástico.

En este caso, se observó una tendencia consistente: a medida que el tamaño de la base de datos disminuye, la métrica 'Mean Metric' tiende a aumentar. Sin embargo, es crucial destacar que estos incrementos son lineales y no excesivamente pronunciados. Esta observación es indicativa de que, incluso con una reducción en el volumen de los datos, nuestra metodología sigue siendo eficiente.

La naturaleza lineal de este incremento refuerza la idea de que la capacidad predictiva del modelo no se ve comprometida en gran medida por la disminución del tamaño de los datos, lo cual es un hallazgo significativo para la aplicación práctica del modelo en situaciones con disponibilidad limitada de datos.

Un hallazgo particularmente intrigante surge en el escenario determinista con solo 5 evaluaciones por componente. En este caso, contrariamente a la tendencia general observada, se registra una disminución en la métrica 'Mean Metric'. Este resultado inesperado podría ser una manifestación de la naturaleza adaptable y estocástica de las redes neuronales. Es posible que, durante el proceso de búsqueda y optimización llevado a cabo por Optuna, se haya identificado una configuración de red neuronal óptima para esta cantidad específica de datos. Esta configuración óptima podría haber contribuido a una mejora notable en la métrica para este conjunto de datos en particular, destacando la capacidad de la red para ajustarse y rendir eficientemente incluso con una cantidad limitada de datos.

Más allá de la evaluación de la métrica 'Mean Metric', se realizaron también pruebas estadísticas para estas bases de datos, comparándolas con aquellas de mayor tamaño. En el caso de las trayectorias deterministas, se analizaron los errores relativos promedio en X y en Y, así como el porcentaje de diferencias que superan el 100 %. Estos resultados se presentan en detalle en la Tabla 3.15, y se complementan con una representación gráfica en la Figura 3.4. Estos análisis estadísticos son fundamentales para comprender en profundidad la precisión y fiabilidad del modelo en diferentes escalas de datos, proporcionando una perspectiva más amplia sobre el desempeño del modelo en distintos escenarios de tamaño de datos.

Modelo	**Error X**	**Error Y**	**Dif.** $\geq 100\,\%$ **X**	**Dif.** $\geq 100\,\%$ **Y**
Convolucional 200	12.62 %	14.94 %	1.57 %	3.42 %
Convolucional 100	15.45 %	17.93 %	3.27 %	3.36 %
Convolucional 50	13.48 %	14.30 %	2.74 %	2.66 %
Convolucional 20	13.35 %	14.78 %	3.12 %	2.66 %
Convolucional 10	16.49 %	17.66 %	3.09 %	2.94 %

Tabla 3.15: Análisis estadístico de los errores relativos de trayectorias. Error X, Y representa el error relativo medio de X, Y, mientras que Dif. $\geq 100\,\%$ X,Y el porcentaje de puntos cuyo error relativo es mayor al 100 %.

A pesar de la reducción progresiva en el tamaño de las bases de datos, desde las más grandes con 200 evaluaciones temporales hasta las más pequeñas con 10 evaluaciones, no se ha detectado un incremento significativo en los errores promedios ni en las diferencias extremas.

Los errores promedio en las coordenadas X e Y (Error X y Error Y) muestran un leve incremento al disminuir el número de evaluaciones temporales en los modelos, lo que indica una pérdida de precisión moderada. No obstante, el aumento no es desproporcionado y se mantiene dentro de un rango que se considera manejable para los propósitos de esta investigación.

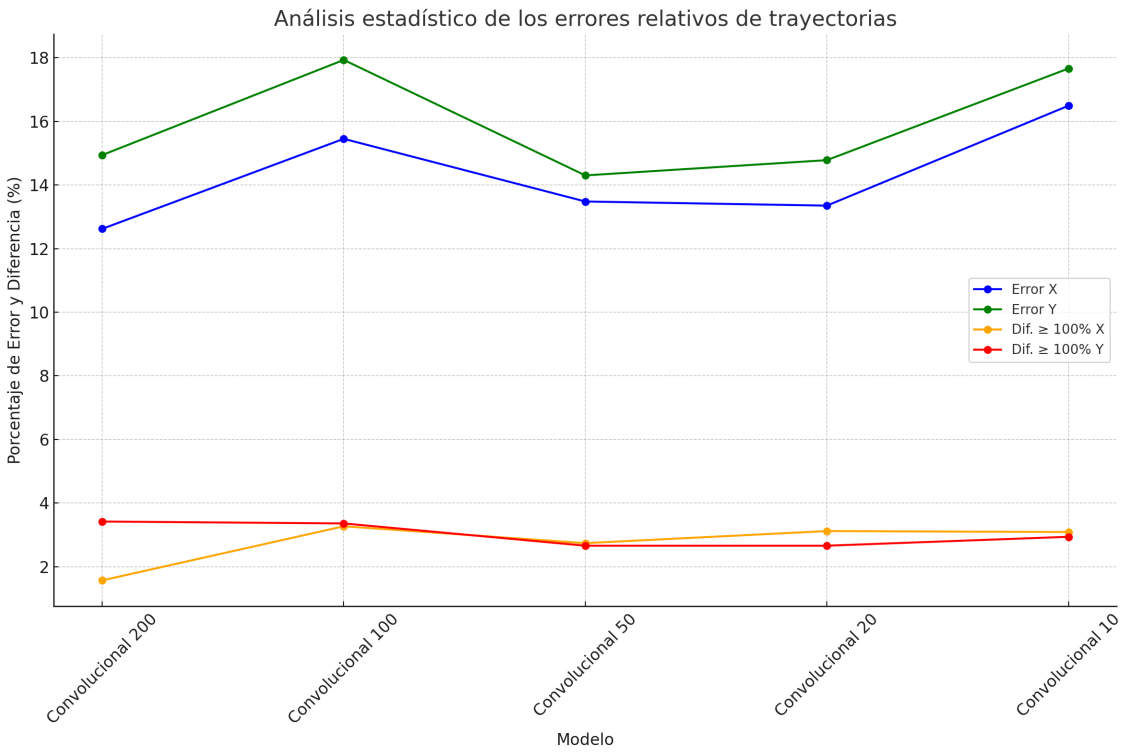


Figura 3.4: Análisis estadístico de los errores relativos de trayectorias en modelos deterministas. Se presentan los errores relativos medios en X (azul) y Y (verde), así como el porcentaje de puntos con errores relativos superiores al 100 % en X (naranja) y Y (rojo) para modelos que varían de 200 a 10 unidades.

El análisis de las diferencias que exceden el 100 % revela que, aunque hay un aumento mínimo en la frecuencia de estas a medida que las bases de datos se hacen más pequeñas, el porcentaje sigue siendo bajo. Esto sugiere que los casos de errores extremadamente altos son poco frecuentes, incluso en las bases de datos más reducidas.

El análisis de los errores relativos en modelos estocásticos revela tendencias importantes relacionadas con la disminución de las dimensiones de las bases de datos. De acuerdo con la Tabla 3.16 y su representación gráfica en la Figura 3.5, que muestra la evolución de los errores relativos promedios y el porcentaje de diferencias mayores al 100 % en función del tamaño de la base de datos, se observan incrementos en los errores al pasar de bases de datos más grandes a más pequeñas.

Modelo	**Error X**	**Error Y**	**Dif. $\geq$ 100 % X**	**Dif. $\geq$ 100 % Y**
Convolucional 200	14.18 %	14.46 %	2.87 %	2.00 %
Convolucional 100	14.70 %	15.67 %	3.52 %	1.68 %
Convolucional 50	13.72 %	14.79 %	1.98 %	2.54 %
Convolucional 20	18.09 %	18.02 %	5.49 %	3.8 %
Convolucional 10	20.56 %	20.88 %	5.51 %	5.07 %

Tabla 3.16: Análisis estadístico de los modelos para trayectorias estocásticas

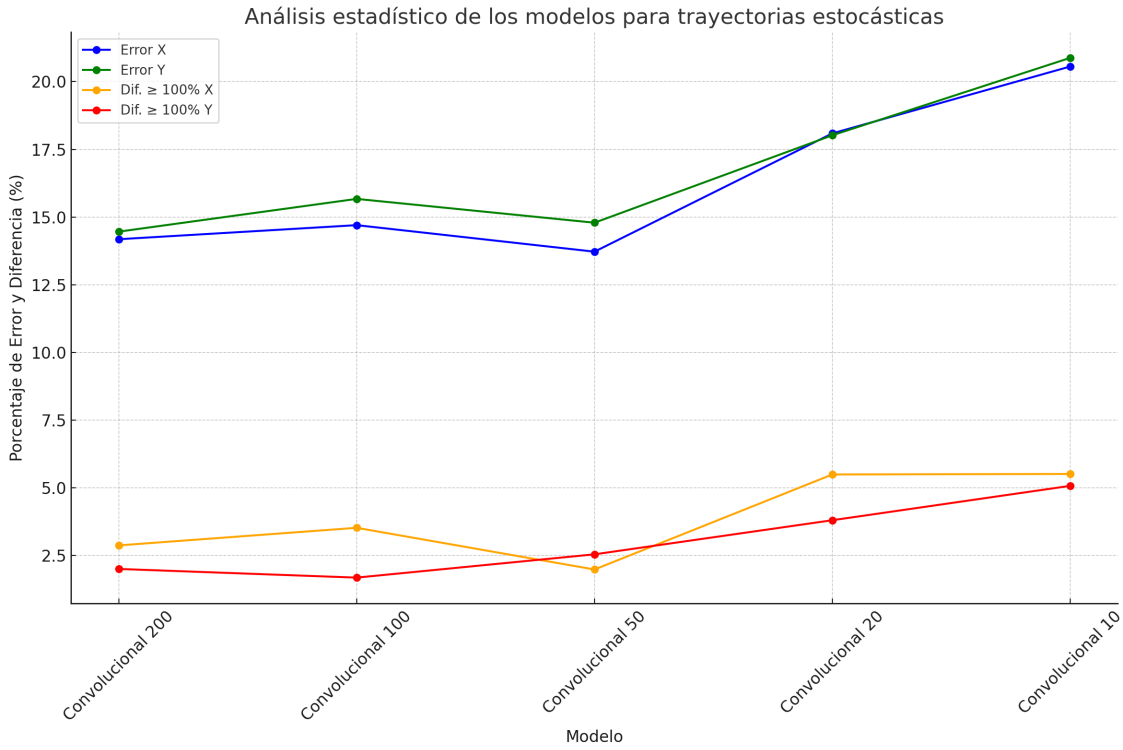


Figura 3.5: Análisis estadístico de errores en modelos convolucionales para trayectorias estocásticas. La gráfica muestra los errores relativos medios en X (azul) y Y (verde), así como el porcentaje de puntos con errores relativos superiores al 100 % en X (naranja) y Y (rojo) para modelos de 200 a 10 unidades.

Aunque existe un aumento en los errores promedios tanto para X como para Y con la disminución del tamaño de las bases de datos, este incremento no es desproporcionado. Es más pronunciado en comparación con el caso determinista, pero no compromete significativamente la eficiencia de la metodología utilizada. El porcentaje de diferencias mayores al 100 %, a pesar de un aumento notable en las bases de datos más pequeñas, particularmente en la coordenada Y para el modelo con solo 10 evaluaciones, permanece en un rango controlable.

Este fenómeno subraya la importancia de una selección cuidadosa en el tamaño de la base de datos para modelos estocásticos, ya que, aunque la metodología sigue siendo robusta, los efectos de la reducción son más evidentes en comparación con el entorno determinista.

Para el caso de los campos vectoriales, su análisis estadístico se puede observar en la Tabla 3.17, y su representación gráfica en la Figura 3.6 tanto para los modelos deterministas como estocásticos.

Modelo	**Porcentajes Deterministas**	**Porcentajes Estocásticos**
Convolucional 100	4.72 %	11.40 %
Convolucional 50	4.58 %	8.23 %
Convolucional 20	2.87 %	13.70 %
Convolucional 10	13.52 %	25.70 %
Convolucional 5	5.53 %	33.13 %

Tabla 3.17: Análisis estadístico de los errores relativos de campos vectoriales

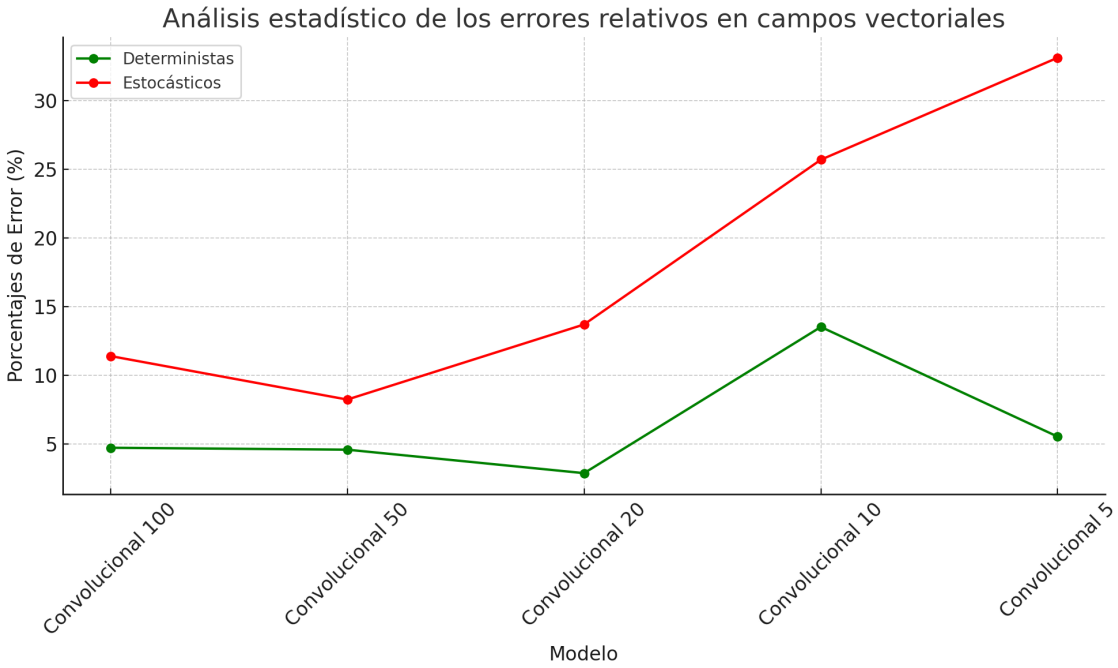


Figura 3.6: Análisis estadístico de los errores relativos en campos vectoriales. Para los datos deterministas, se observa que el porcentaje del error se mantiene constante a lo largo de las diferentes dimensiones de los campos, mientras que para los datos estocásticos, vemos un incremento lineal conforme disminuye la dimensión del campo vectorial.

Los modelos deterministas muestran un ligero incremento en el error relativo a medida que el tamaño de la base de datos disminuye, lo cual es un testimonio de la robustez de la metodología aplicada. A pesar de la reducción en la cantidad de datos, los resultados se mantienen en un rango aceptable, lo que implica que la metodología sigue siendo eficiente.

En comparación, los modelos estocásticos evidencian una sensibilidad mayor a la escasez de datos, reflejada en un aumento más significativo del error relativo. Aunque esta tendencia es más pronunciada, los resultados no indican un deterioro sustancial de la precisión, y los modelos conservan su utilidad práctica.

De manera interesante, se observa en los modelos deterministas con solo 5 evaluaciones una disminución en el error relativo, lo cual podría indicar que, para ciertas configuraciones específicas, la metodología es capaz de ofrecer resultados precisos incluso con una cantidad muy limitada de datos. Este resultado adicional podría ser producto de una arquitectura de red neuronal optimizada por Optuna, aprovechando la estocasticidad inherente a las redes neuronales.

En conjunto, estos hallazgos subrayan la eficacia general de la metodología en distintos tamaños de bases de datos y resaltan la capacidad de Optuna para encontrar configuraciones óptimas bajo ciertas condiciones.

Capítulo 4

Conclusión

En el capítulo precedente, realizamos un análisis profundo y detallado de los resultados alcanzados a través de la implementación de variados modelos de redes neuronales, enfocados en la identificación precisa de parámetros dentro del sistema de regulación genética del Toggle Switch. Este análisis abarcó escenarios tanto deterministas como estocásticos. En el presente capítulo, nos proponemos sintetizar estos hallazgos para extraer conclusiones significativas que resalten la trascendencia y el impacto de esta investigación en el campo de la biología de sistemas y la ingeniería genética. Nuestro objetivo es ofrecer una visión clara de cómo estos resultados contribuyen a la comprensión y el avance del conocimiento en esta área de estudio.

Aptitud de las Redes Neuronales para Sistemas de Alta Dimensionalidad y No Lineales.

Un hallazgo crucial de nuestra investigación es la idoneidad de las redes neuronales para modelar y analizar sistemas de regulación genética que son inherentemente de alta dimensionalidad y no lineales. La generalización y flexibilidad de las redes neuronales en problemas inversos, como en nuestro estudio del Toggle Switch, demuestra su potencial para ser aplicadas en una variedad más amplia de sistemas biológicos. Estas redes tienen la capacidad de aprender patrones y relaciones complejas que son esenciales no solo para nuestro sistema específico sino que también son aplicables a otros sistemas biológicos, gracias a su habilidad para manejar interacciones no lineales, efectos de retroalimentación y dinámicas temporales.

Además, la efectividad de las redes neuronales en resolver problemas inversos —deducir parámetros y procesos internos a partir de observaciones— es de gran relevancia para una variedad de sistemas biológicos. Esto se ve reforzado por la literatura existente, donde la aplicación de redes neuronales en contextos similares ha demostrado resultados prometedores. Aunque nuestro estudio se enfoca específicamente en el Toggle Switch, los principios y técnicas empleados ofrecen perspectivas valiosas para la investigación en otros sistemas biológicos complejos.

Esta capacidad de las redes neuronales para adaptarse y aprender de sistemas complejos no solo representa un avance técnico, sino también un paso significativo hacia una comprensión más profunda y una manipulación más eficaz de sistemas biológicos en aplicaciones prácticas y de investigación. La investigación presentada en esta tesis destaca el inmenso potencial de las redes neuronales en la biología de sistemas y sugiere nuevas vías de exploración e innovación en este campo interdisciplinario.

Mejora de Hiperparámetros con Optuna.

La implementación de Optuna ha demostrado ser un factor crucial en la mejora del rendimiento de los modelos de redes neuronales. Esta optimización de hiperparámetros contribuye significativamente a la precisión de la predicción en diferentes escenarios, reafirmando la importancia de una selección cuidadosa y metódica de hiperparámetros en modelos de aprendizaje profundo. La mejora observada abarca tanto escenarios deterministas como estocásticos, lo que indica una robustez y adaptabilidad notable de los modelos frente a variaciones en los datos.

Superioridad de las Redes Convolucionales.

Las redes convolucionales han superado consistentemente a las redes densas en términos de precisión y rendimiento. Esta ventaja se atribuye a la habilidad de las redes convolucionales para capturar características espaciales y temporales complejas, lo que resulta crucial en el contexto de la modelación de sistemas biológicos dinámicos como el Toggle Switch. Esta observación es particularmente relevante para futuras investigaciones en biología sintética y modelado de sistemas biológicos, donde las redes convolucionales podrían ofrecer ventajas significativas.

Retos en la Predicción de Parámetros Específicos.

Los modelos, en general, enfrentan mayores desafíos al predecir con precisión los coeficientes a_1, a_2 y n Esta tendencia se puede atribuir a la complejidad inherente de estos parámetros en el sistema de ecuaciones del modelo Toggle Switch (Ecuación 2.1). En este modelo, los coeficentes a_1 y a_2 epresentan las tasas de producción de las proteínas X e Y, respectivamente, y son cruciales para determinar la velocidad y la eficiencia con la que se activan o desactivan los genes en el sistema. Por otro lado, el exponente n refleja la cooperatividad en la regulación genética, un elemento clave para entender cómo las interacciones entre los genes afectan la estabilidad y la dinámica del sistema Toggle Switch. La importancia y la complejidad de estos parámetros radican en su papel en la modulación de la respuesta del sistema a los cambios en las concentraciones de proteínas, lo que resulta esencial para el funcionamiento adecuado del Toggle Switch en aplicaciones de biología sintética.

La dificultad en predecir estos parámetros sugiere que capturan aspectos esenciales y sutiles de la interacción genética, destacando la necesidad de un enfoque más refinado y posiblemente más especializado en el modelado y la interpretación de estos elementos dentro de los sistemas de regulación genética.

Robustez ante Datos con Ruido.

La efectividad de los modelos propuestos para manejar datos con ruido representa un avance significativo en la identificación de parámetros en sistemas de regulación genética. Esta capacidad es fundamental, ya que los sistemas biológicos en el mundo real a menudo están sujetos a variaciones y ruido inherentes, lo que puede desafiar o incluso impedir la aplicación de métodos tradicionales como el de mínimos cuadrados [93] y otras técnicas de identificación de parámetros [94].

En la práctica, el ruido en los datos biológicos puede provenir de múltiples fuentes, incluyendo variaciones experimentales, fluctuaciones ambientales y la naturaleza estocástica de los procesos biológicos a nivel celular y molecular. Métodos convencionales como los mínimos cuadrados, aunque robustos bajo ciertas condiciones, a menudo se ven limitados al tratar con datos ruidosos,

ya que pueden resultar en soluciones poco precisas o directamente en la incapacidad de encontrar una solución viable.

En contraste, los modelos de redes neuronales que hemos desarrollado y evaluado demuestran una notable resistencia y adaptabilidad al ruido. Esta robustez no solo refuerza la precisión de las predicciones en condiciones menos ideales, sino que también resalta la relevancia y la aplicabilidad práctica de estos modelos en la investigación biológica. Pueden proporcionar insights valiosos y predicciones fiables incluso cuando los datos disponibles están lejos de ser perfectos o están sujetos a variaciones impredecibles

Por lo tanto, la capacidad de estos modelos para manejar eficazmente datos ruidosos no solo es un logro técnico importante, sino que también abre nuevas vías para la exploración y el análisis en biología de sistemas y biología sintética, donde la precisión y la fiabilidad son cruciales para avanzar en nuestro entendimiento y manipulación de sistemas biológicos complejos.

Limitaciones de la Función de Costo Personalizada.

Contrariamente a lo que se podría esperar, la personalización de la función de costo no mejoró los resultados de la predicción. Este hallazgo es significativo, ya que desafía la noción de que las funciones de costo personalizadas siempre son beneficiosas. En nuestro estudio, las funciones de costo estándar resultaron ser más adecuadas, lo que resalta la importancia de probar y validar diferentes enfoques en el modelado de redes neuronales.

Influencia del Tamaño del Experimento en las Predicciones.

Los análisis detallados de las métricas 'Mean Metric' y los errores relativos de trayectorias y campos vectoriales subrayan la capacidad de los modelos para mantener una alta precisión a pesar de la reducción en el tamaño de la base de datos. Esta observación es coherente tanto para enfoques deterministas como estocásticos, lo que destaca la escalabilidad y la robustez de la metodología propuesta.

En el contexto de los modelos deterministas, incluso con una base de datos reducida a 5 evaluaciones, la metodología no solo se mantuvo eficiente sino que, en ciertos casos, mostró una mejora en la precisión. Esto puede interpretarse como una confirmación de la habilidad del optimizador Optuna para descubrir arquitecturas de red que maximizan la eficacia del modelo bajo restricciones de datos.

Los modelos estocásticos, aunque mostraron un aumento en el error relativo con bases de datos más pequeñas, no evidenciaron un deterioro significativo en la precisión, lo que refuerza la pertinencia de la metodología en escenarios de datos variados, comunes en la investigación biológica donde los conjuntos de datos pueden fluctuar en volumen.

Estos hallazgos resaltan la importancia de las herramientas de optimización como Optuna en la mejora de la eficiencia de las redes neuronales, especialmente en la regulación genética y otros campos biotecnológicos donde la obtención de grandes volúmenes de datos puede ser desafiante.

Modelo único ineficiente.

A lo largo de este estudio, se puso a prueba la hipótesis de que un modelo único podría adaptarse de manera eficiente a diferentes tamaños de bases de datos sin necesidad de

ajustar su arquitectura. Sin embargo, los resultados obtenidos indican que esta hipótesis no se sostiene. La ineficiencia del modelo único se hizo evidente a través de una serie de métricas y pruebas estadísticas que reflejaron una disminución en la precisión y fiabilidad de las predicciones.

Los resultados del modelo único, particularmente para los campos vectoriales, estuvieron por debajo de las expectativas. A pesar de la flexibilidad teórica que ofrecía la arquitectura propuesta, en la práctica, el modelo no logró capturar la complejidad inherente a los datos con la eficacia que se preveía.

Una de las posibles explicaciones para estos resultados puede radicar en la necesidad de un ajuste más fino de los hiperparámetros o en la adaptación de la estructura de la red a las peculiaridades de cada conjunto de datos. Asimismo, el modelo único puede haberse visto limitado por la falta de capacidad para especializarse y ajustarse a las características específicas presentes en cada tamaño de datos. Esta observación abre un campo de indagación sobre la importancia de la personalización de modelos en función de las características únicas de cada conjunto de datos analizado.

Las conclusiones obtenidas del análisis del modelo único subrayan la necesidad de continuar explorando estrategias de modelado más sofisticadas. Estos resultados también recalcan la importancia de validar los modelos de aprendizaje automático no solo en términos de su rendimiento con conjuntos de datos ideales, sino también en su aplicabilidad a datos reales, con todas sus imperfecciones y desafíos.

Direcciones futuras.

En futuras investigaciones, sería recomendable explorar modelos híbridos o enfoques de ensamble que puedan combinar las fortalezas de diferentes arquitecturas para mejorar la precisión y robustez. Además, se podría investigar el impacto de una selección más detallada de hiperparámetros y la incorporación de técnicas de regularización más avanzadas para mejorar la generalización de los modelos. Este aprendizaje subraya el valor de la iteración y la experimentación continua en el campo del aprendizaje automático aplicado a sistemas biológicos complejos.

A pesar de que el tamaño de la base de datos no influyó sustancialmente en los resultados de este estudio, es esencial continuar explorando la escalabilidad de los modelos. Investigaciones futuras deberían centrarse en determinar el tamaño óptimo de datos necesario para obtener predicciones robustas y precisas, así como en evaluar el rendimiento de la metodología en diferentes configuraciones experimentales y tipos de datos. Este enfoque no solo mejorará la comprensión de las limitaciones y capacidades de los modelos actuales, sino que también guiará el desarrollo de nuevas estrategias para el manejo eficiente de datos en la investigación científica.

La investigación presentada en esta tesis ha demostrado de manera concluyente la eficacia y versatilidad de los modelos de redes neuronales en la identificación precisa de parámetros en sistemas de regulación genética. La adaptabilidad de estos modelos a diferentes tipos de datos y su notable robustez frente a la variabilidad y el ruido inherentes a los sistemas biológicos, destacan su idoneidad para aplicaciones avanzadas en biología sintética y el análisis de sistemas de regulación genética. Este trabajo no solo resalta el inmenso potencial de las redes neuronales en la biología de sistemas, sino que también marca un hito importante para futuras investigaciones que aspiren a integrar el aprendizaje automático con modelos biológicos complejos. Los resultados obtenidos y las metodologías desarrolladas en este estudio son un testimonio del éxito de esta tesis, abriendo caminos para nuevas indagaciones y aplicaciones en este fascinante campo interdisciplinario.

Apéndice A

Forma intuitiva del Algoritmo del Descenso de Gradiente

Supongamos que intentamos minimizar alguna función C(v), que puede ser cualquier función de varias variables ($v = v_1, v_2, \dots$). Nos podría ayudar a visualizarlo tomando a C como una función de dos variables que llamaremos v_1 y v_2 como en la Figura A.1. Lo que nos gustaría es encontrar dónde C alcanza su mínimo global.

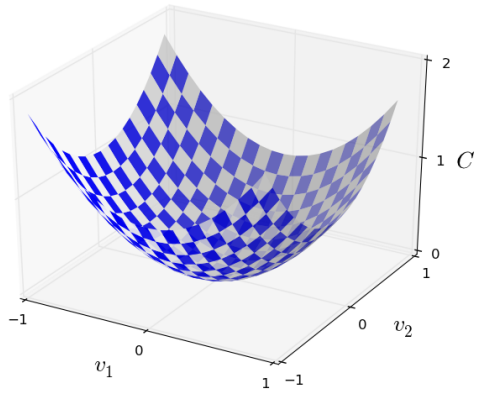


Figura A.1: Función $C(v_1, v_2)$ dependiente de dos variables.

Una forma de atacar el problema es utilizar el cálculo para tratar de encontrar el mínimo analíticamente. Podríamos calcular derivadas y luego intentar usarlas para encontrar lugares donde C es un extremo. Con un poco de suerte, eso podría funcionar cuando C es una función de solo una o unas pocas variables. Pero se complicará cuando tengamos muchas más variables. Y para las redes neuronales, a menudo queremos muchas más variables: las redes neuronales más grandes tienen funciones de costo que dependen de miles de millones de pesos y sesgos de una manera extremadamente complicada.

Debido a que el cálculo no nos sirve en este problema tendremos que atacarlo de otra manera. Empezamos pensando en nuestra función como una especie de valle e imaginamos una

pelota rodando por la ladera de nuestro valle. Nuestra experiencia diaria nos dice que la bola eventualmente rodará hasta el fondo. Elegiríamos al azar un punto de partida para una bola (imaginaria) y luego simularíamos el movimiento de la bola mientras rueda hacia el fondo del valle.

Para hacer esta pregunta más precisa, pensemos en lo que sucede cuando movemos la pelota una pequeña cantidad Δv_1 en la dirección v_1 y una pequeña cantidad Δv_2 en la dirección v_2. C cambia de la siguiente manera:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 \tag{A.1}$$

Lo que haremos será escoger valores Δv_1 y Δv_2 tal que hagan ΔC negativa. Para saber como hacer esa elección nos ayudará a definir Δv como un vector cuyas componentes serán los cambios en sus direcciones, $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$, en donde T se refiere a la traspuesta del vector. También definiremos el gradiente de C como el vector de las derivadas parciales, $\left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$, y denotaremos al gradiente de C como ∇C:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T \tag{A.2}$$

Con estas definiciones, la ecuación (A.1) para ΔC puede ser escrita como:

$$\Delta C \approx \nabla C \cdot \Delta v \tag{A.3}$$

Esta ecuación nos permite ver cómo elegir Δv para hacer que ΔC sea negativo. Imaginemos que escogemos:

$$\Delta v = -\eta \nabla C \tag{A.4}$$

Donde η es un parámetro positivo y pequeño que llamaremos tasa de aprendizaje. Metiendo esto en la ecuación (A.3), nos queda que:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2 \tag{A.5}$$

Ahora, debido a que $||\nabla C|| \geq 0$, esto garantiza que $\Delta C \leq 0$, por lo que C siempre decrecerá si tomamos v como en la ecuación (A.4). Por lo tanto usaremos dicha ecuación para calcular el valor de Δv, y con esto movemos la posición v de la pelota por la cantidad:

$$v \to v' = v - \eta \nabla C \tag{A.6}$$

Luego usaremos esta misma regla de nuevo para hacer un nuevo movimiento. Si seguimos haciendo esto, una y otra vez, seguiremos disminuyendo C hasta que, esperamos, alcancemos un mínimo global.

En resumen, la forma en que funciona el algoritmo de descenso de gradiente es calcular repetidamente el gradiente ∇C, y luego moverse en la dirección opuesta, cayendo por la pendiente del valle.

Hemos visto el algoritmo de descenso de gradiente para cuando C es una función de solo 2 variables, pero todo funciona igual cuando C es una función de más variables. Para el caso en que C es una función de m variables $v_1, \ldots, v_m$ el cambio ΔC en C producido por un pequeño cambio $\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$ es:

$$\Delta C \approx \nabla C \cdot \Delta v \tag{A.7}$$

En donde el gradiente ∇C es el vector:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)^T \tag{A.8}$$

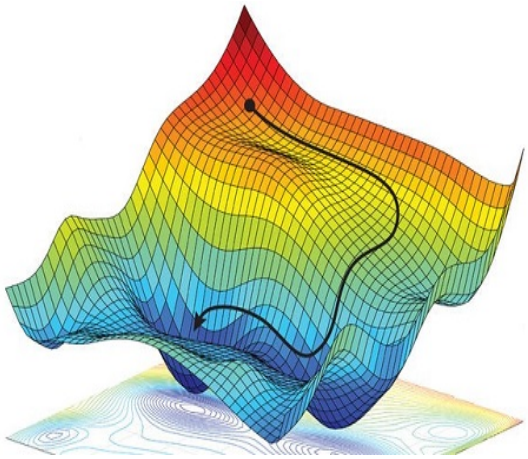


Figura A.2: Visualización gráfica del algoritmo del descenso de gradiente.

Y así como en el caso de dos variables, podemos elegir:

$$\Delta v = -\eta \nabla C \tag{A.9}$$

Esto nos da una forma de seguir el gradiente al mínimo, incluso cuando C es una función de muchas variables, aplicando repetidamente la regla de actualización:

$$v \to v' = v - \eta \nabla C \tag{A.10}$$

Apéndice B

Ejemplo numérico Algoritmo Backpropagation

Para empezar imaginemos que tenemos una pequeña red como la que sigue:

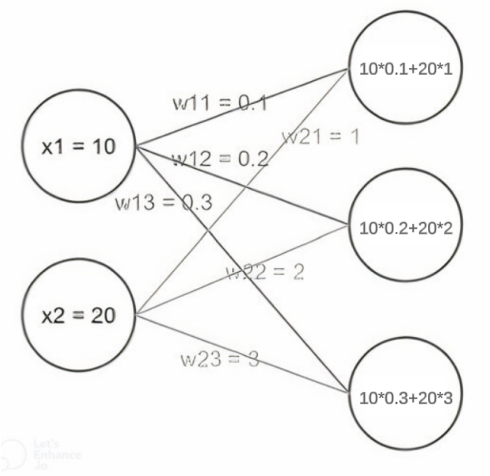


Figura B.1: Red neuronal pequeña con pesos indicados

Por ejemplo, si queremos representar un peso w que conecte con la segunda neurona en la capa 1 con la tercera neurona en la capa 2, lo haremos con un peso que llamaremos w_{23} y en este caso ese peso es de 3. Si queremos hacer esto con todos los pesos podemos escribir una matrix de pesos que conecte las dos capas:

$$\begin{bmatrix} w_{11}(=0{,}1) & w_{12}(=0{,}2) & w_{13}(=0{,}3) \\ w_{21}(=1) & w_{22}(=2) & w_{23}(=3) \end{bmatrix} \tag{B.1}$$

Llamaremos a esta matriz $\mathbf{w}$. Si tenemos nuestra matriz de entrada como:

$$\begin{bmatrix} x_1 = 10 \\ x_2 = 20 \end{bmatrix} \tag{B.2}$$

Podemos hacer la multiplicación $\mathbf{w}^T\mathbf{x}$ para obtener una matriz 3×1 que daría como resultado la salida, que llamándolo el vector $\mathbf{z}$, sería:

$$\mathbf{z} = (21, 42, 63)^T \tag{B.3}$$

Con esto hemos descrito, junto con la estructura de las neuronas y las conexiones, el paso de las entradas a través de la red que se denomina forward pass.

El forward pass es simplemente la suma de los cálculos que ocurren cuando la entrada viaja a través de la red neuronal. Podemos ver cada capa como una función de los datos que le llegan, por lo que si $\mathbf{x}$ es el vector de entrada, $\mathbf{y}$ es el vector de la capa de salida y f_i,f_h, y f_0 son las funciones generales calculadas en cada capa respectivamente, podemos decir que el vector de salida es de la forma:

$$\mathbf{y} = f_0(f_h(f_i(\mathbf{x}))) \tag{B.4}$$

Esta forma de ver a las redes neuronales es muy importante para corregir sus pesos a través del algoritmo de backpropagation.

Recordemos que el objetivo de una red neuronal es encontrar un buen conjunto de pesos y sesgos, y esto se hace mediante el entrenamiento a través de backpropagation hacia atrás, que es lo contrario del forward pass. La idea es medir el error que comete la red al clasificar y luego modificar el peso para que este error sea muy pequeño. Recordemos que un perceptrón toma decisiones, tal que:

$$salida = \begin{cases} 0 & \text{si } w \cdot x + b \leq 0 \\ 1 & \text{si } w \cdot x + b > 0 \end{cases}$$

En donde $w \cdot x \equiv \sum_j w_j x_j$. Si tomamos a $b + \sum_j w_j x_j = z$ y a $y = salida$, podemos ver de la siguiente figura que exista una manera de absorber el sesgo como uno de los pesos, por lo que lo único que necesitemos al momento de que aprenda nuestra red es una regla de actualización de los pesos.

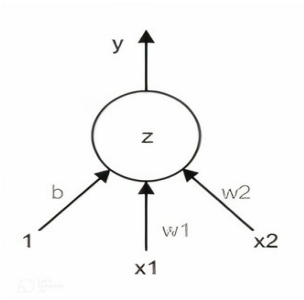


Figura B.2: Ilustración de un perceptrón individual con dos entradas y un sesgo: El sesgo se representa como una entrada adicional con valor constante 1 y su respectivo peso b, que se ajusta durante el entrenamiento de la red.

Para absorber el sesgo como un peso se necesita incluir una entrada x_0 con el valor 1 y el sesgo es el peso de dicha entrada. Esto se puede escribir como:

$$z = b + \sum_j w_j x_j = w_0 x_0 (= b) + w_1 x_1 + w_2 x_2 + \ldots \tag{B.5}$$

Como podemos ver, b puede ser tanto x_0 como w_0 mientras que el otro sea 1. Así que si queremos cambiar el sesgo con el aprendizaje y el entretenimiento y las entradas nunca cambian por obvias razones, necesitamos tratar al sesgo como un peso. A este proceso se le denomina "bias absorption"(absorción de sesgos).

Ahora, imaginémonos que compramos comida una vez al día para nuestro desayuno en algún supermercado. Todos los días nuestra comida consta de una pieza de pollo, unas calabazas

hervidas y cierta cantidad de arroz. El cajero solo nos da la cantidad total, que varía cada día. Supongamos que el precio de los componentes no varía mucho con el tiempo y lo que queremos hacer es saber cuanto cuesta el kg de pollo, de calabazas y de arroz. Podemos medir el peso de cada uno, pero no el precio de cada uno, solo la suma total de todo. Tenga en cuenta que una comida no será suficiente para deducir los precios, ya que tenemos tres y no sabemos qué componente es responsable de la proporción del precio total.

Podemos observar que el precio por kilogramo es en realidad similar al peso de la red neuronal. Para ver esto, piense en cómo hallaría el precio por kilogramo de los componentes de la comida: adivinando los precios por kilogramo de los componentes, multiplicando por el peso de cada producto y comparando su suma con el precio que realmente se pagó. Si por ejemplo el precio total fue de $60, podemos pensar que por cada uno se pudo pagar $20 y ajustar los costos por kilogramo para que de esto, por ejemplo si compramos 200 gr de pollo el precio por kilogramo sería de $100. Podríamos haber pensado de igual manera que cada componente costó 30, 20 y 10 pesos respectivamente pero de todas formas tendríamos que esperar a la siguiente ida al supermercado para volver a comprar los mismos productos y ver si estos costos son los correctos. Por supuesto, deseamos corregir las estimaciones para que nuestro error sea cada vez menos a medida que pasan las comidas y, con suerte, esto nos llevará a una buena aproximación.

Vemos que existe un precio real por kilogramo aunque no lo sepamos y nuestro método es tratar de descubrirlo calculando cuánto fallamos en el total con nuestras predicciones. Esto es principalmente lo que hacemos en una red neuronal, para que una vez que hallamos una buena aproximación podremos calcular con una buena exactitud el precio total de nuestras futuras comidas aún sin haber visto el precio en nuestro ticket.

Matemáticamente, cada comida tiene la siguiente forma:

$$\begin{aligned} total =& ppk_{pollo} \cdot cant_{pollo} + ppk_{calabaza} \cdot cant_{calabaza} \\ &+ ppk_{arroz} \cdot cant_{arroz} \end{aligned} \tag{B.6}$$

Donde total es el precio total, cant es la cantidad de cada alimento y ppk es el precio por kilogramo de cada componente. Cada comida tiene un precio total que sabemos y las cantidades que sabemos, por lo que cada comida toma el valor de una constricción de los pesos por kilogramo(ppk).

Imaginémonos que el precio real de cada cosa es de $ppk_{pollo} = 100$, $ppk_{calabaza} = 30$ y $ppk_{arroz} = 50$, y empezamos intentando con $ppk_{pollo} = 60$, $ppk_{calabaza} = 30$ y $ppk_{arroz} = 30$, sabiendo que compramos 0.23 kg de pollo, 0.15 kg de calabaza y 0.27 kg de arroz, y que pagamos 30 pesos en total. Multiplicando los precios que adivinamos con las cantidades que compramos obtenemos 13.8, 4.5 y 8.1 que nos da en total 26.4 pesos, siendo 3.5 pesos menor al valor real. Este valor es llamado el error residual e intentaremos minimizarlo en futuras iteraciones (comidas), por lo que necesitamos distribuirlo en nuestros ppk´s. Hacemos esto cambiando nuestros ppk´s mediante:

$$\Delta ppk_i = \frac{1}{n} \cdot cant_i(t - y) \tag{B.7}$$

Donde $i \in \{pollo, calabaza, arroz\}$, n es el número de elementos, en este caso 3, $quant_i$ es la cantidad de i, t es el precio total y y es el precio predicho total. Esto es conocido como la regla delta (delta rule). Escribiendo esto en el lenguaje de las redes neuronales toma la forma:

$$\Delta w_i = \eta x_i(t - y) \tag{B.8}$$

En donde w_i son los pesos, x_i son las entradas y de nuevo $t - y$ es el error residual. En este caso, como sabemos η es conocido como la tasa de aprendizaje. En este caso el valor default que

escogimos fue $1/n$.

Con esto en mente, podemos recordar a nuestra función sigmoide:

$$y = \frac{1}{1+e^{-z}} \tag{B.9}$$

Y que para calcular si el resultado de nuestras iteraciones va mejorando o empeorando usábamos el error cuadrático medio:

$$E = \frac{1}{2}\sum_n \left(t^{(n)} - y^{(n)}\right)^2 \tag{B.10}$$

donde $t^{(n)}$ denota el objetivo de nuestro dato de entrenamiento n, mientras que $y^{(n)}$ es nuestra predicción. Para ver como varía nuestra función de costo (el error cuadrático medio) cuando cambia alguna de nuestras w_i derivamos la expresión anterior con respecto a w_i:

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\sum_n \frac{\partial y^{(n)}}{\partial w_i}\frac{dE^{(n)}}{dy^{(n)}} \tag{B.11}$$

Para ello empezamos a derivar usando la regla de la cadena. Primero vemos que:

$$\frac{\partial z}{\partial w_i} = \frac{\partial \sum_i w_i x_i}{\partial w_i} = x_i \tag{B.12}$$

Despúes tenemos que derivar $\frac{dy}{dz}$, para ello vemos que:

$$\frac{dy}{dz}\left(\frac{1}{1+e^{-z}}\right) = \frac{e^{-z}}{(1+e^{-z})^2} \tag{B.13}$$

Pero podemos ver que:

$$\frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\cdot\frac{e^{-z}}{1+e^{-z}} \tag{B.14}$$

Vemos que el primer término es precisamente y mientras que el segundo podemos ver que:

$$\frac{e^{-z}}{1+e^{-z}} = \frac{(1+e^{-z})-1}{1+e^{-z}} = \frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}} \tag{B.15}$$

$$= 1 - \frac{1}{1+e^{-z}} = 1 - y \tag{B.16}$$

Por lo que llegamos a:

$$\frac{dy}{dz} = y(1-y) \tag{B.17}$$

Por lo que con esto tenemos que por la regla de la cadena:

$$\frac{\partial y}{\partial w_i} = x_i y(1-y) \tag{B.18}$$

Ahora si ya podemos hacer $\frac{dE}{dy}$ que es nuestro último paso para la regla de la cadena. Tenemos que:

$$\frac{dE}{dy}\left[\frac{1}{2}(t-y)^2\right] = -(t-y) \tag{B.19}$$

por lo tanto:

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^{(n)}}{\partial w_i}\frac{\partial E}{\partial y^{(n)}} \tag{B.20}$$

$$= -\sum_n x_i^{(n)} y^{(n)}(1-y^{(n)})(t^{(n)} - y^{(n)}) \tag{B.21}$$

Como podemos ver esto es muy similar a la regla delta para la neurona lineal, sin embargo tiene términos extra $y^{(n)}(1-y^{(n)})$ que es la parte de la función sigmoide.

Con todo esto que hemos visto, sabemos ahora como se utilizan las derivadas para aprender los pesos de una neurona logística, y sin saberlo hemos estado aprendiendo sobre el algoritmo de backpropagation, ya que es casi lo mismo que hemos estado haciendo, aplicando más de una vez para ir hacia atrás en los errores a través de las capas de neuronas.

Hay que recordar además el algoritmo del descenso de gradiente, ya que backpropagation es básicamente ese algoritmo aplicado. Matemáticamente backpropagation es:

$$\omega_{actualizado} = \omega_{viejo} - \eta \nabla E \tag{B.22}$$

Donde ω es el peso, η es la tasa de entrenamiento y E es la función de costo que mide el desempeño general.

Nos preguntamos si se puede hacer el aprendizaje de los pesos de una manera más simple, sin usar derivadas ni el descenso del gradiente. Podemos utilizar la siguiente aproximación: seleccionamos un peso ω y lo modificamos un poco para ver si eso ayuda. Si lo hace nos quedamos con el cambio, si hace que las cosas funcionen peor, el cambio lo hacemos en dirección contraria (en vez de añadir el pequeño cambio al peso, lo restamos). Después, si esto hace las cosas mejores nos quedamos con este cambio, sin embargo si tampoco las hace mejores, podemos concluir que dicho ω es perfecto y nos movemos al siguiente peso.

Sin embargo existen varios problemas con el proceso antes mencionado. Primero, el proceso toma mucho más tiempo, ya que después de que cambiamos el peso, necesitamos hacer por lo menos un par de pruebas para ver si este cambio mejoró o empeoró el proceso. En segundo lugar, cuando cambiamos los pesos individuales no podemos saber si una combinación diferente de ellos podría hacer un proceso mejor, por ejemplo si tenemos dos pesos ω y v y hacemos pequeños cambios en ellos por separado, puede pasar que el proceso empeore, sin embargo si se hace junto puede hacer las cosas mejores. Estos dos problemas se pueden solucionar con el descenso de gradiente, sin embargo el segundo parcialmente, ya que podemos caer a mínimos locales de la función y no a un mínimo general.

El tercero de los problemas es el ya conocido que al final del entrenamiento, los cambios deberán de ser demasiados pequeños y esto podría hacer que tarde nuestro proceso de aprendizaje. Sin embargo, backpropagation también tiene dicho problema, sin embargo es usualmente resuelto utilizando una taza de entrenamiento que cambia con el tiempo, haciéndose más pequeña a medida que se lleva a cabo el proceso de aprendizaje.

Volvamos al algoritmo de backpropagation. Examinemos que sucede en las capas ocultas de una red neuronal feedforward. Empezamos con pesos y sesgos aleatorios multiplíquelos con las entradas, súmelos y llévelos a través de la regresión logística que los aplana a un valor entre 0 y 1, y lo hacemos una vez más. Al final, tenemos un valor entre 0 y 1 de la neurona logística de la capa de salida. Podemos decir que cualquier cosa mayor a 0.5 es un 1 y menor un 0, pero si la red nos da 0.67 y la salida debió ser 0 , lo único que sabemos es el error que nos produce la red (a la que ya hemos llamado E) y que podemos aprovechar. Queremos medir como E cambia cuando ω_i cambia, lo que significa que queremos encontrar la derivada de E con respecto a las actividades de la capa oculta. Queremos encontrar todas las derivadas al mismo tiempo y para esto usamos vectores y matrices y por lo mismo el gradiente. Una vez que tengamos las derivadas de E con respecto a las capas ocultas, fácilmente podremos saber los cambios por los pesos solamente.

Imaginemos que tenemos una red simple con una sola capa oculta con una neurona en cada capa. Usaremos el subíndice s para la capa de salida y o para la capa oculta. Recordemos que tenemos que nuestra función de error es:

$$E = \frac{1}{2} \sum_{s \in salida} (t_s - y_s)^2 \tag{B.23}$$

Lo primero que debemos hacer es convertir la diferencia entre la salida y el valor objetivo en una derivación de error. Como ya lo habíamos visto:

$$\frac{\partial E}{\partial y_s} = -(t_s - y_s) \tag{B.24}$$

Ahora, necesitamos formular la derivada del error con respecto de y_s a un error con respecto a z_s. Para esto usamos la regla de la cadena:

$$\frac{\partial E}{\partial z_s} = \frac{\partial y_s}{\partial z_s} \frac{\partial E}{\partial y_s} = y_s(1 - y_s) \frac{\partial E}{\partial y_s} \tag{B.25}$$

Ahora podemos calcular la derivada del error con respecto a y_o:

$$\frac{\partial E}{\partial y_o} = \sum_s \frac{dz_s}{dy_o} \frac{\partial E}{\partial z_s} = \sum_s w_{os} \frac{\partial E}{\partial z_s} \tag{B.26}$$

Estos pasos que hemos hecho de $\frac{\partial E}{\partial y_s}$ a $\frac{\partial E}{\partial y_o}$ son el corazón de backpropagation, ya que se puede repetir esto para ir hacia atrás a través de las capas cuantas veces queramos. Una vez teniendo $\frac{\partial E}{\partial z_s}$, podemos ver que:

$$\frac{\partial E}{\partial w_o s} = \frac{\partial z_s}{\partial w_o s} \frac{\partial E}{\partial z_s} = y_i \frac{\partial E}{\partial z_j} \tag{B.27}$$

A partir de aquí la regla para actualizar los pesos es bastante sencilla, y la llamamos la regla general de actualización del peso:

$$w_i^{nuevo} = w_i^{viejo} + (-1)\eta \frac{\partial E}{\partial w_i^{viejo}} \tag{B.28}$$

Donde de nuevo η es la tasa de aprendizaje, mientras que el factor de -1 es debido a que buscamos minimizar E, si lo dejáramos positivo lo estaríamos maximizando. En forma vectorial queda como:

$$w_i^{nuevo} = w_i^{viejo} - \eta \nabla E \tag{B.29}$$

Para la tasa de aprendizaje η existen varias posibilidades:

1. η está fijo y no cambia con el tiempo.

2. η cambia con el tiempo pero es única para todos los pesos.

3. η cambia para cada conexión de capas.

Como ejemplo, podemos ver la siguiente red :

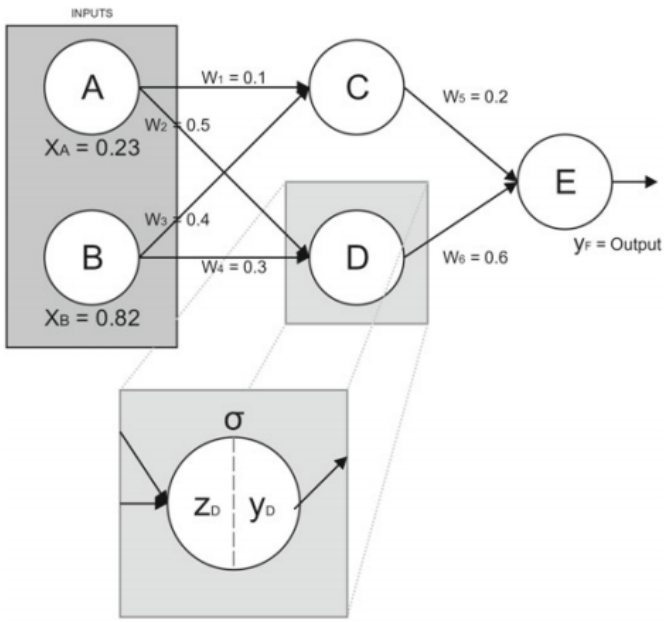


Figura B.3: Diagrama de una red neuronal sencilla, mostrando las conexiones ponderadas entre las neuronas de entrada (A y B) y la capa de salida (E) a través de una capa oculta (C y D), con la función de activación (σ) aplicada a las salidas de las neuronas.

Podemos asumir que todas las neuronas de nuestra red tienen una función de activación logística y los pesos y las entradas que indica en la imagen, así que necesitamos primero correr la red, después aplicar backpropagation para volver a correr y ver el decaimiento del error.

Podemos ver en nuestra red que tenemos dos neuronas de entrada. La neurona A tiene como entrada $x_A = 0{,}23$ y la neurona B tiene como entrada $x_b = 0{,}82$. El objetivo para esta entrada de entrenamiento (que consiste en x_A u x_B) es que la salida sea 1. La capa de salida y la capa oculta tiene la siguiente función de activación:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{B.30}$$

Empezamos calculando el primer ciclo (anteriormente lo habíamos llamado época). El primer paso es calcular las salidas de las neuronas C y D, nombradas como y_C y y_D respectivamente. Se calculan mediante:

$$y_C = \sigma(0{,}23 \cdot 0{,}1 + 0{,}82 \cdot 0{,}4) = \sigma(0{,}351) = 0{,}5868$$
$$y_D = \sigma(0{,}23 \cdot 0{,}5 + 0{,}82 \cdot 0{,}3) = \sigma(0{,}361) = 0{,}5892$$

Y ahora usamos estas salidas como entradas de la neurona E, que nos da el resultado final:

$$y_E = \sigma(0{,}5868 \cdot 0{,}2 + 0{,}5892 \cdot 0{,}6) = \sigma(0{,}4708) = 0{,}6155$$

Ya con esto podemos calcular el error, recordando que usamos la función error cuadrático medio. Así que ponemos en el objetivo 1 y en la salida 0.6155, para que nos dé:

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}(1 - 0{,}6155)^2 = 0{,}0739$$

Empecemos recalculando los pesos w_5 y w_3 ya que los otros pesos se calculan de manera similar. Como ya sabemos backpropagation va de la última neurona a la primera, por lo que calcular w_5 es más sencillo, por lo que lo haremos primero. Para ello primero necesitamos saber como un cambio en w_5 afecta E y queremos hacer esos cambios tal que minimicen la función de costo. Para ello necesitamos calcular:

$$\frac{\partial E}{\partial \omega_5} = \frac{\partial E}{\partial y_E} \cdot \frac{\partial y_E}{\partial z_E} \cdot \frac{\partial z_E}{\partial \omega_5}$$

Que como podemos ver, son derivadas que ya hemos sacado con anterioridad. Veamos que pasa en este caso con $\frac{\partial E}{\partial y_E}$:

$$\frac{\partial E}{\partial y_E} = -(t - y_E)$$

Como ya lo habíamos sacado con anterioridad; con los datos que tenemos, nos da que:

$$\frac{\partial E}{\partial y_E} = -(1 - 0{,}6155) = -0{,}3844$$

Mientras que para el caso de $\frac{\partial y_E}{\partial z_E}$, tenemos que:

$$\frac{\partial y_E}{\partial z_E} = y_E(1 - y_E)$$

Que con los datos que tenemos queda como:

$$\frac{\partial y_E}{\partial z_E} = 0{,}6155(1 - 0{,}6155) = 0{,}2365$$

Por lo que lo único que nos falta por calcular es $\frac{\partial z_E}{\partial w_5}$. Tenemos que:

$$\begin{aligned} z_E &= y_C \cdot w_5 + y_D \cdot w_6 \\ \Rightarrow \frac{\partial z_E}{\partial w_5} &= y_C \cdot 1 y_D \cdot 0 = y_C = 0{,}5868 \end{aligned}$$

Tomando estos valores de nuevo para la regla de la cadena que habíamos ocupado para calcular $\frac{\partial E}{\partial \omega_5}$, nos queda:

$$\begin{aligned} \frac{\partial E}{\partial \omega_5} &= \frac{\partial E}{\partial y_E} \cdot \frac{\partial y_E}{\partial z_E} \cdot \frac{\partial z_E}{\partial \omega_5} = -0{,}3844 \cdot 0{,}2365 \cdot 0{,}5868 \\ &= -0{,}0533 \end{aligned}$$

Repetimos el mismo proceso para obtener $\frac{\partial E}{\partial w_6} = -0{,}0535$ (el único cambio aqúi sería en la parte de $\frac{\partial z_E}{\partial w_5}$, en donde en este caso al hacer la derivada nos daría 0 para w_5 y 1 para w_6). Ahora lo único que hace falta es usar estos valores en la regla general para la actualización de los pesos que ya habíamos discutido antes (recordamos que el valor η es un valor que puede ser tanto variable como constante, aquí lo pusimos como un valor constante de 0.7):

$$\begin{aligned} w_5^{nuevo} &= w_5^{viejo} - \eta \frac{\partial E}{\partial w_5} = 0{,}2 - (0{,}7 \cdot (-0{,}0533)) \\ &= 0{,}2373 \end{aligned}$$

Mientras que:

$$w_6^{nuevo} = 0{,}6374$$

Ahora continuamos con la siguiente capa, sin embargo hay algo importante que hacer notar aquí: como podemos ver, necesitamos los valores de ω_5 y ω_6 para encontrar los siguientes pesos

$\omega_1, \omega_2, \omega_3$ y ω_4 y tenemos que utilizar los valores viejos no los nuevos que acabamos de sacar. Cuando tengamos todos los valores actualizados es cuando correremos de nuevo toda la red.

Procedamos ahora a la capa oculta. Lo que necesitamos ahora es actualizar el peso w_3. Podemos ver de la figura (B.3) que para llegar de la neurona de salida a w_3 necesitamos pasar a través de C, por lo tanto:

$$\frac{\partial E}{\partial \omega_3} = \frac{\partial E}{\partial y_C} \cdot \frac{\partial y_C}{\partial z_C} \cdot \frac{\partial z_C}{\partial \omega_3}$$

Empezamos viendo que:

$$\frac{\partial E}{\partial y_C} = \frac{\partial z_E}{\partial y_C}\frac{\partial E}{\partial z_E} = \omega_5 \frac{\partial E}{\partial z_E} = \omega_5 \frac{\partial y_E}{\partial z_E} \cdot \frac{\partial E}{\partial y_E}$$

Pero como podemos ver, ya habíamos sacado estas derivadas parciales con anterioridad, por lo tanto:

$$\frac{\partial E}{\partial y_C} = 0{,}2 \cdot 0{,}2365 \cdot (-0{,}3844) = 0{,}2 \cdot (-0{,}0909) = -0{,}0181$$

Mientras tanto para $\frac{\partial y_C}{\partial z_C}$:

$$\frac{\partial y_C}{\partial z_C} = y_C(1 - y_C) = 0{,}5868 \cdot (1 - 0{,}5868) = 0{,}2424$$

Además necesitamos $\frac{\partial z_C}{\partial \omega_3}$; recordando que $z_C = x_1 \cdot \omega_1 + x_2 \cdot \omega_2$, nos queda que:

$$\frac{\partial z_C}{\partial \omega_3} = x_1 \cdot 0 + x_2 \cdot 1 = x_2 = 0{,}82$$

Con todo esto ya tenemos:

$$\begin{aligned}\frac{\partial E}{\partial \omega_3} &= \frac{\partial E}{\partial y_C} \cdot \frac{\partial y_C}{\partial z_C} \cdot \frac{\partial z_C}{\partial \omega_3} = -0{,}0181 \cdot 0{,}2424 \cdot 0{,}82 \\ &= -0{,}0035\end{aligned}$$

Por lo que usando la regla de actualización de pesos:

$$\omega_3^{nuevo} = 0{,}4 - (0{,}7 \cdot (-0{,}0035)) = 0{,}4024$$

Utilizando los mismos pasos podemos encontrar $\omega_1^{nuevo} = 0{,}1007$. Mientras tanto para llegar a ω_2^{nuevo} y ω_4^{nuevo} necesitamos ir a través de D. Por lo tanto nos queda que:

$$\frac{\partial E}{\partial \omega_3} = \frac{\partial E}{\partial y_D} \cdot \frac{\partial y_D}{\partial z_D} \cdot \frac{\partial z_D}{\partial \omega_3}$$

Ya sabemos el procedimiento:

$$\begin{aligned}\frac{\partial E}{\partial y_D} &= \omega_6 \cdot \frac{\partial E}{\partial z_F} = 0{,}6 \cdot (-0{,}0909) = -0{,}0545 \\ \frac{\partial y_C}{\partial z_C} &= y_D(1 - y_D) = 0{,}5892(1 - 0{,}5892) = 0{,}2420\end{aligned}$$

Además:

$$\begin{aligned}\frac{\partial z_D}{\partial \omega_2} &= 0{,}23 \\ \frac{\partial z_D}{\partial \omega_4} &= 0{,}82\end{aligned}$$

Ya con esto tenemos que:

$$\omega_2^{nuevo} = 0{,}5 - 0{,}7 \cdot (-0{,}0545 \cdot 0{,}2420 \cdot 0{,}23) = 0{,}502$$
$$\omega_4^{nuevo} = 0{,}3 - 0{,}7 \cdot (-0{,}0545 \cdot 0{,}2420 \cdot 0{,}82 = 0{,}307$$

Y con esto acabamos. Haciendo un resumen tenemos que:

$$\omega_1^{nuevo} = 0{,}1007$$
$$\omega_2^{nuevo} = 0{,}507$$
$$\omega_3^{nuevo} = 0{,}4024$$
$$\omega_4^{nuevo} = 0{,}307$$
$$\omega_5^{nuevo} = 0{,}2373$$
$$\omega_6^{nuevo} = 0{,}6374$$
$$E^{viejo} = 0{,}0739$$

Con estos datos podemos correr la red para corroborar que los nuevos pesos hacen que el error decreció:

$$y_C^{nuevo} = \sigma(0{,}23 \cdot 0{,}1007 + 0{,}82 \cdot 0{,}4024) = \sigma(0{,}3531) = 0{,}5873$$
$$y_D^{nuevo} = 0{,}5907$$
$$y_F^{nuevo} = \sigma(0{,}5873 \cdot 0{,}2373 + 0{,}5907 \cdot 0{,}4024) = \sigma(0{,}5158) = 0{,}6261$$
$$E^{nuevo} = \frac{1}{2}(1 - 06261)^2 = 0{,}0699$$

Que comparándolo con el E_{viejo}, vemos que el error ha disminuido. Recordemos que en este proceso solo pusimos a prueba una entrada de entrenamiento, el vector (0.23,0.82). Sin embargo es posible usar varias pruebas de entrenamiento para generar el error y encontrar el gradiente (recordemos que a esto lo habíamos llamado un mini-batch) y podemos hacer esto varias veces y por cada repetición haremos una iteración (de nuevo recordemos que a esto lo habíamos llamado una epoch).

Con esto entendemos ahora lo que hicimos en nuestra red neuronal pasada, sin embargo esta red es completamente hecha a mano, y actualmente existen librerías que pueden hacer este trabajo en unas cuantas líneas de código como veremos a continuación.

Apéndice C

Implementación de redes neuronales con Keras y Tensorflow

Esta sección presenta dos modelos de redes neuronales, uno denso y otro convolucional, diseñados específicamente para analizar los datos de los campos vectoriales de 20 evaluaciones. A continuación, se detalla la configuración del modelo de red neuronal densa:

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Dropout
from keras.optimizers import Adam
from keras.regularizers import l1, l2

model = Sequential()

model.add(Flatten(input_shape=(2, 20, 20)))  # Tus dimensiones de entrada

model.add(Dense(71, activation='relu', kernel_initializer='HeNormal',
    kernel_regularizer=l2(0.001)))

model.add(Dense(77, activation='relu', kernel_initializer='random_normal',
    kernel_regularizer=l2(0.001)))
model.add(Dropout(0.1127))

model.add(Dense(7))
model.summary()
```

En el modelo de red neuronal presentado, se inicia con una capa de entrada de dimensiones (2, 20, 20), seguida de dos capas densas con 71 y 77 neuronas respectivamente. Además, se incluye una capa de Dropout con una tasa de 0.1127 para reducir el sobreajuste. La estructura se completa con una capa de salida personalizada. Para obtener una visión detallada de la arquitectura del modelo, se utiliza el método model.summary(), que proporciona un resumen exhaustivo de las capas, sus formas y el número total de parámetros como se muestra en la Figura C.1

En lo que respecta a las redes neuronales convolucionales, su estructura se presenta de la siguiente forma:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l1, l2

model = Sequential()

model.add(Conv2D(58, (4, 1), activation='relu', input_shape=(20, 20, 2),
    kernel_initializer='random_normal', kernel_regularizer=l1(0.001)))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(28, activation='relu', kernel_initializer='HeNormal',
    kernel_regularizer=l1(0.001)))

model.add(Dense(41, activation='relu', kernel_initializer='HeNormal',
    kernel_regularizer=l1(0.001)))

model.add(Dense(120, activation='relu', kernel_initializer='HeNormal',
    kernel_regularizer=l1(0.001)))

model.add(Dropout(0.3))

model.add(Dense(7))

model.summary()
```

En esta red, comenzamos con una capa convolucional que utiliza 58 filtros, cada uno de tamaño (4,1). Posteriormente, las capas son aplanadas mediante una capa Flatten, seguida de tres capas densas con 28, 41 y 120 neuronas respectivamente. La red concluye con una capa de Dropout con una tasa de 0.3 y una capa densa final de 7 neuronas.

```
Model: "sequential_327"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_324 (Flatten)       (None, 800)               0

 dense_954 (Dense)           (None, 71)                56871

 dense_955 (Dense)           (None, 77)                5544

 dropout_233 (Dropout)       (None, 77)                0

 dense_956 (Dense)           (None, 7)                 546

=================================================================
Total params: 62,961
Trainable params: 62,961
Non-trainable params: 0
_________________________________________________________________
```

Figura C.1: Resumen de la arquitectura de un modelo secuencial en Keras, mostrando las capas, formas de salida y número de parámetros.

Estos ejemplos ilustra la facilidad de implementar redes neuronales usando Keras y TensorFlow, donde se logra una configuración completa con pocas líneas de código. No obstante, el verdadero desafío reside en la comprensión y aplicación adecuada de estas herramientas para resolver problemas específicos. Todos los códigos de las redes desarrolladas en esta investigación están detallados en [92] para una consulta más exhaustiva.

Bibliografía

[1] Drew Endy, *Foundations for engineering biology*, Nature, vol. 438, no. 7067, pp. 449–453, 2005.

[2] Jay D. Keasling, *Synthetic biology for synthetic chemistry*, ACS Chemical Biology, vol. 3, no. 1, pp. 64–76, 2010.

[3] George M. Church, Michael B. Elowitz, Christina D. Smolke, Christopher A. Voigt, and Ron Weiss, *Synthetic biology*, Science, vol. 346, no. 6210, 2014.

[4] Uri Alon, *Network motifs: theory and experimental approaches*, Nature Reviews Genetics, vol. 8, no. 6, pp. 450–461, 2007.

[5] Michael Levine and Eric H. Davidson, *Gene regulatory networks for development*, Proceedings of the National Academy of Sciences, vol. 102, no. 14, pp. 4936–4942, 2005.

[6] Jeff Hasty, David McMillen, and James J. Collins, *Engineered gene circuits*, Nature, vol. 420, no. 6912, pp. 224–230, 2002.

[7] Hiroaki Kitano, *Systems biology: a brief overview*, Science, vol. 295, no. 5560, pp. 1662–1664, 2002.

[8] Leland H. Hartwell, John J. Hopfield, Stanislas Leibler, and Andrew W. Murray, *From molecular to modular cell biology*, Nature, vol. 402, no. C47, pp. C47–C52, 1999.

[9] John J. Tyson, Katherine C. Chen, and Bela Novak, *Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell*, Current Opinion in Cell Biology, vol. 15, no. 2, pp. 221–231, 2003.

[10] Peter S. Swain, Michael B. Elowitz, and Eric D. Siggia, *Intrinsic and extrinsic contributions to stochasticity in gene expression*, Proceedings of the National Academy of Sciences, vol. 99, no. 20, pp. 12795–12800, 2002.

[11] Michael B. Elowitz, Arnold J. Levine, Eric D. Siggia, and Peter S. Swain, *Stochastic gene expression in a single cell*, Science, vol. 297, no. 5584, pp. 1183–1186, 2002.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016.

[13] James J. Collins, *Synthetic biology: integrating biology with engineering*, IEEE Engineering in Medicine and Biology Magazine, vol. 29, no. 3, pp. 12–18, 2012.

[14] Roger D. King and Stephen G. Oliver, *The future of biotechnology*, Nature, vol. 525, no. 7567, pp. 65–67, 2015.

[15] M. Peifer and S. Timmer, *Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting*, IET Systems Biology, vol. 1, no. 2, pp. 78–88, 2007.

[16] E. O. Voit, *Computational Analysis of Biochemical Systems*, Cambridge University Press, 2006.

[17] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, *Using Bayesian networks to analyze expression data*, Journal of Computational Biology, vol. 7, no. 3-4, pp. 601–620, 2000.

[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[19] N. Metropolis and S. Ulam, *The Monte Carlo method*, Journal of the American Statistical Association, vol. 44, no. 247, pp. 335–341, 1949.

[20] D. T. Gillespie, *Exact stochastic simulation of coupled chemical reactions*, The Journal of Physical Chemistry, vol. 81, no. 25, pp. 2340–2361, 1996.

[21] Kitano, H. (2002). Systems biology: a brief overview. Science, 295(5560), 1662-1664.

[22] Alon, U. (2007). An introduction to systems biology: design principles of biological circuits. CRC press.

[23] Karr, J. R., Sanghvi, J. C., Macklin, D. N., Gutschow, M. V., Jacobs, J. M., Bolival Jr, B., ... & Covert, M. W. (2012). A whole-cell computational model predicts phenotype from genotype. Cell, 150(2), 389-401.

[24] Yamamoto, K., Watanabe, H., & Ishihama, A. (2014). Expression levels of transcription factors in Escherichia coli: growth phase- and growth condition-dependent variation of 90 regulators from six families. *Microbiology*, 160(9), 1903-1913.

[25] Barabási, A. L., Gulbahce, N., & Loscalzo, J. (2011). Network medicine: a network-based approach to human disease. Nature Reviews Genetics, 12(1), 56-68.

[26] Estrada, E., & Rodriguez-Velazquez, J. A. (2020). Understanding complex systems: A network science perspective. In Handbook of Statistics (Vol. 40, pp. 45-73). Elsevier.

[27] Huang, S., Chaudhary, K., & Garmire, L. X. (2018). More is better: recent progress in multi-omics data integration methods. Frontiers in Genetics, 9, 84.

[28] Kang, H. J., Kawasawa, Y. I., Cheng, F., Zhu, Y., Xu, X., Li, M., & Soares, M. J. (2011). Spatio-temporal transcriptome of the human brain. Nature, 478(7370), 483-489.

[29] Ravindra, V. M., Patel, A., & Cohen, J. V. (2020). Dynamic changes in the immune microenvironment in melanoma following targeted and immune therapies. In Seminars in Cancer Biology (Vol. 65, pp. 125-134). Academic Press.

[30] Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. Journal of theoretical biology, 22(3), 437-467.

[31] Wilkinson, D. J. (2011). Stochastic modelling for systems biology. CRC press.

[32] Hill, A. V. (1910). The Possible Effects of the Aggregation of the Molecules of Haemoglobin on its Dissociation Curves. Journal of Physiology, 40(4), iv-vii.

[33] Monod, J., & Jacob, F. (1965). General conclusions: Teleonomic mechanisms in cellular metabolism, growth, and differentiation. Cold Spring Harbor Symposia on Quantitative Biology, 26, 389-401.

[34] Bintu, L., Buchler, N. E., Garcia, H. G., Gerland, U., Hwa, T., Kondev, J., & Phillips, R. (2005). Transcriptional regulation by the numbers: applications. Current opinion in genetics & development, 15(2), 125-135.

[35] Sneppen, K., Pedersen, S., Krishna, S., Dodd, I., & Semsey, S. (2010). Simplified models of biological networks. Annual Review of Biophysics, 39, 43-59.

[36] Gardner, T. S., Cantor, C. R., & Collins, J. J. (2000). Construction of a genetic toggle switch in Escherichia coli. Nature, 403(6767), 339-342.

[37] Elowitz, M. B., & Leibler, S. (2000). A synthetic oscillatory network of transcriptional regulators. Nature, 403(6767), 335-338.

[38] Weber, W., & Fussenegger, M. (2007). Genetic switches: systems biology and drug discovery. Drug discovery today, 12(9-10), 408-415.

[39] Strogatz, S. H. (2014). Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering. CRC Press.

[40] Sheldon M. Ross, *Introduction to probability models*, Academic press, 2014.

[41] Athanasios Papoulis, S. Unnikrishna Pillai, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, 2002.

[42] Geoffrey R. Grimmett, David R. Stirzaker, *Probability and Random Processes*, Oxford University Press, 2001.

[43] Norris, J. R. (1997). *Markov Chains*. Cambridge University Press.

[44] Gardiner, C. (2009). *Stochastic Methods: A Handbook for the Natural and Social Sciences*. Springer.

[45] Kloeden, P. E., & Platen, E. (1999). *Numerical Solution of Stochastic Differential Equations*. Springer.

[46] J. Paulsson, *Summing up the noise in gene networks*, Nature, 427(6973), 415-418, 2004.

[47] A. Eldar, M. B. Elowitz, *Functional roles for noise in genetic circuits*, Nature, 467(7312), 167-173, 2010.

[48] M. Thattai, A. van Oudenaarden, *Intrinsic noise in gene regulatory networks*, Proceedings of the National Academy of Sciences, 98(15), 8614-8619, 2001.

[49] T. B. Kepler, T. C. Elston, *Stochasticity in transcriptional regulation: Origins, consequences, and mathematical representations*, Biophysical Journal, 81(6), 3116-3136, 2001.

[50] Dormand, J. R., & Prince, P. J. (1980). A family of embedded Runge-Kutta formulae. Journal of computational and applied mathematics, 6(1), 19-26.

[51] Hairer, E., Nørsett, S. P., & Wanner, G. (1993). Solving Ordinary Differential Equations I: Nonstiff Problems. Springer Series in Computational Mathematics.

[52] Higham, D. J. (2001). An algorithmic introduction to numerical simulation of stochastic differential equations. SIAM review, 43(3), 525-546.

[53] Tarantola, A. (2005). Inverse problem theory and methods for model parameter estimation. Society for Industrial and Applied Mathematics.

[54] Aster, R. C., Borchers, B., & Thurber, C. H. (2018). Parameter estimation and inverse problems. Elsevier.

[55] Hansen, P. C. (2010). Discrete Inverse Problems: Insight and Algorithms. SIAM.

[56] Russell, S., Norvig, P., Davis, E., & Frew, J. (2009). Artificial Intelligence: A Modern Approach. Pearson.

[57] Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.

[58] Bengio, Yoshua. 2016. Deep Learning. Adaptive Computation and Machine Learning Series. London, England: MIT Press.

[59] Haykin, S. (2009). Neural Networks and Learning Machines. Pearson.

[60] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4), 115-133.

[61] Kandel, Eric R and Schwartz, James H and Jessell, Thomas M and Siegelbaum, Steven A and Hudspeth, A James and Mack, Sarah, *Principles of neural science*, McGraw-hill New York, 2000.

[62] Hebb, Donald Olding, *The organization of behavior: a neuropsychological theory*, Psychology Press, 1949.

[63] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.

[64] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386-408.

[65] Minsky, M., & Papert, S. (1969). Perceptrons: An Introduction to Computational Geometry. MIT Press.

[66] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

[67] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[68] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural Networks, 12(1), 145-151.

[69] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12, 2121-2159.

[70] Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.

[71] Kingma, D.P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[72] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.

[73] Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

[74] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[75] Brillinger, D. R. (1981). Time series: Data analysis and theory. Holden-Day.

[76] Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285.

[77] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

[78] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (pp. 1097-1105).

[79] Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer.

[80] Oliphant, Travis E. "Python for Scientific Computing." Computing in Science & Engineering, vol. 9, no. 3, 2007, pp. 10-20.

[81] Van Rossum, Guido and Drake, Fred L. "Python reference manual". PythonLabs, Virginia, USA, 1999.

[82] Pilgrim, Mark. "Dive Into Python 3." Apress, 2009.

[83] Oliphant, Travis E. "A guide to NumPy." Trelgol Publishing USA, 2006.

[84] Jones, Eric, Oliphant, Travis, Peterson, Pearu and others. "SciPy: Open source scientific tools for Python." 2001.

[85] McKinney, Wes. "Data Structures for Statistical Computing in Python." Proceedings of the 9th Python in Science Conference, 2010, pp. 51 - 56.

[86] Abadi, Martín and others. "TensorFlow: Large-scale machine learning on heterogeneous systems." 2016, Software available from tensorflow.org.

[87] Chollet, François and others. "Keras." 2015, Available at keras.io.

[88] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, y Masanori Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework*. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, páginas 2623–2631, 2019. ACM.

[89] Strogatz, S.H. (2014). *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Westview Press; 2nd edition.

[90] Perko, L. (2001). *Differential Equations and Dynamical Systems*. Springer; 3rd edition.

[91] Collette, A. (2013). *Python and HDF5*. O'Reilly Media.

[92] Sánchez, C. (2023). *Tesis-Maestria*. GitHub. https://github.com/RafaSanCed/Tesis-Maestria.git

[93] Smith, J. (2018). *Handling Noise in Biological Data*. Journal of Computational Biology, 25(6), 653-664.

[94] Jones, M. (2019). *Advanced Methods for Parameter Identification in Biological Systems*. Systems Biology, 12(3), 234-245.