UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

APLICACIÓN DE TÉCNICAS DE INTELIGENCIA ARTIFICIAL PARA LA PREDICCIÓN EFICIENTE DEL BALANCE DE FLUJO METABÓLICO CELULAR, Y SU POSTERIOR INTEGRACIÓN EN EL SIMULADOR GRO

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN TELECOMUNICACIONES, SISTEMAS Y ELECTRÓNICA

PRESENTA:

RICARDO JIMÉNEZ BENÍTEZ

ASESOR:

ING. JOSÉ LUIS BARBOSA PACHECO

CUAUTITLÁN IZCALLI, ESTADO DE MÉXICO, 2020

VNIVERSIDAD NACIONAL AVPNOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

DEPARTAMENTO DE EXAMENES PROFESIONALES

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

Aplicación de técnicas de inteligencia artificial para la predicción eficiente del balance de flujo metabólico celular, y su posterior integración en el simulador GRO.

Que presenta el pasante: **Ricardo Jiménez Benítez**
Con número de cuenta: **414070019** para obtener el título de: **Ingeniero en Telecomunicaciones, Sistemas y Electrónica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 23 de Septiembre de 2020.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Mtro. José Isaac Sánchez Guerra	
1er. SUPLENTE	Dr. David Tinoco Varela	
2do. SUPLENTE	Mtra. Judith Mayte Flores Pérez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMCF/javg

VNIVERSIDAD NACIONAL AVPNPMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

DEPARTAMENTO DE EXAMENES PROFESIONALES

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

Aplicación de técnicas de inteligencia artificial para la predicción eficiente del balance de flujo metabólico celular, y su posterior integración en el simulador GRO.

Que presenta el pasante: **Ricardo Jiménez Benítez**
Con número de cuenta: **414070019** para obtener el título de: **Ingeniero en Telecomunicaciones, Sistemas y Electrónica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO.**

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 23 de Septiembre de 2020.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Mtro. José Isaac Sánchez Guerra	
1er. SUPLENTE	Dr. David Tinoco Varela	
2do. SUPLENTE	Mtra. Judith Mayte Flores Pérez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMCF/javg

VNIVERSIDAD NACIONAL AVTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

DEPARTAMENTO DE EXAMENES PROFESIONALES

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

Aplicación de técnicas de inteligencia artificial para la predicción eficiente del balance de flujo metabólico celular, y su posterior integración en el simulador GRO.

Que presenta el pasante: **Ricardo Jiménez Benítez**
Con número de cuenta: **414070019** para obtener el título de: **Ingeniero en Telecomunicaciones, Sistemas y Electrónica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO.**

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 23 de Septiembre de 2020.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Mtro. José Isaac Sánchez Guerra	
1er. SUPLENTE	Dr. David Tinoco Varela	
2do. SUPLENTE	Mtra. Judith Mayte Flores Pérez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMCF/javg

VNIVERSIDAD NACIONAL AVPNOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales de la FES Cuautitlán.

DEPARTAMENTO DE EXAMENES PROFESIONALES

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

Aplicación de técnicas de inteligencia artificial para la predicción eficiente del balance de flujo metabólico celular, y su posterior integración en el simulador GRO.

Que presenta el pasante: **Ricardo Jiménez Benítez**
Con número de cuenta: **414070019** para obtener el título de: **Ingeniero en Telecomunicaciones, Sistemas y Electrónica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO.**

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 23 de Septiembre de 2020.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Mtro. José Isaac Sánchez Guerra	
1er. SUPLENTE	Dr. David Tinoco Varela	
2do. SUPLENTE	Mtra. Judith Mayte Flores Pérez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMCF/javg

VNIVERSIDAD NACIONAL AVTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

DEPARTAMENTO DE EXAMENES PROFESIONALES

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

Aplicación de técnicas de inteligencia artificial para la predicción eficiente del balance de flujo metabólico celular, y su posterior integración en el simulador GRO.

Que presenta el pasante: **Ricardo Jiménez Benítez**
Con número de cuenta: **414070019** para obtener el título de: **Ingeniero en Telecomunicaciones, Sistemas y Electrónica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 23 de Septiembre de 2020.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Mtro. José Isaac Sánchez Guerra	
1er. SUPLENTE	Dr. David Tinoco Varela	
2do. SUPLENTE	Mtra. Judith Mayte Flores Pérez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMGF/ivm

AGRADECIMIENTOS

Este trabajo de tesis es parte del esfuerzo, dedicación y compromiso como parte de mi desarrollo profesional. Expreso mi más sincero agradecimiento a todas aquellas personas que han sido parte de mi carrera como estudiante en los diferentes niveles educativos y que con su apoyo moral, económico y psicológico han hecho parte de mi desarrollo personal.

Agradezco especialmente a mi madre, mi padre y mi hermana por siempre estar en la unión como familia para enfrentar los retos de la vida. Gracias por sus concejos, educación, apoyo moral, la perseverancia y el amor en todo momento, he podido crecer como persona. Gracias por inculcarme los valores del respeto, solidaridad, disciplina y humildad para ejercer mi carrera profesional. Por siempre seguir los principios del esfuerzo, determinación, fortaleza, seguridad y la fe, puede lograr uno lo que se proponga. Por lo anterior y mucho más, gracias.

Agradezco respetuosamente al profesor José Luis Barbosa Pacheco porque además de ser el asesor de esta tesis, siempre nos ha brindado su apoyo moral y profesional para crecer como individuos a nivel personal y académico. Por motivarnos a dar siempre lo mejor de nosotros, a pesar de los obstáculos que se presenten, siempre encontrando el aspecto positivo de las cosas y sacar lo mejor de ello, sabiendo que cada día se puede aprender algo nuevo con esfuerzo, dedicación y disciplina.

Agradezco a la Universidad Nacional Autónoma de México por ser impulsora del desarrollo de científico, cultural y educativo en México. Gracias a los académicos por brindarnos sus conocimientos he impulsarnos a desarrollar de forma ética nuestra carrera profesional. Hago una mención especial al departamento de la DGECI por impulsar a los estudiantes a compartir y aprender de las culturas alrededor del mundo, además porque gracias a su apoyo y confianza he podido llevar a cabo la estancia de investigación.

De igual manera, agradezco cordialmente a la Universidad Politécnica de Madrid, al grupo de trabajo del Laboratorio de Inteligencia Artificial y al profesor Alfonso Rodríguez-Patón por permitirme realizar la estancia de investigación, brindarme su apoyo y conocimiento durante la realización de este trabajo. Gracias a ellos por sus contribuciones, ideas y conocimiento durante el desarrollo de este trabajo.

Agradezco a mis amigos por su cariño, apoyo y los agrandes momentos durante mi carrera universitaria. Los grandes proyectos, la convivencia y el compartir la mesa, se comparte con los grandes amigos. La amistad no conoce fronteras. Mi más admiración, cariño y respeto a todos ellos.

Por último, quiero agradecer al grupo de investigadores y autores que han sido citados en este trabajo que, con su esfuerzo y dedicación han proporcionado el conocimiento teórico y práctico para el desarrollo de este trabajo de tesis.

“Por mi raza hablará el espíritu”

RESUMEN

La tecnología avanza constantemente y un claro ejemplo de ello es el desarrollo de la Inteligencia Artificial aplicada al campo de la biología de sistemas. Concretamente las Redes Neuronales y los Algoritmos Genéticos destacan como técnicas aplicadas a la búsqueda de patrones y la automatización de cálculos matemáticos que por otros medios resultarían ineficientes.

Los modelos basados en agentes aplicados en simuladores computacionales se presentan como campo de estudio y aplicación de la biología de sistemas, enfocada a modelar agentes bio-químicos a través de la reconstrucción en escala genómica de la red metabólica.

GRO es un simulador sintético multicelular de circuitos genéticos basado en agentes, que cumple la función de emular crecimiento poblacional, en virtud del modelo metabólico de cada agente celular de forma individual. Debido a que GRO calcula el metabolismo de cada individuo dentro de un ambiente de crecimiento poblacional de miles de agentes, hace ineficiente la simulación en agentes bacterianos como el *E. coli*, ya que integran diversidad de parámetros, provocando un mayor costo computacional.

Este trabajo de investigación propone minimizar el costo computacional que se requiere para simular el crecimiento poblacional de agentes microbianos mediante el entrenamiento de Redes Neuronales de retro propagación. Como en la actualidad no se cuenta con un conjunto de entrenamiento basado en el metabolismo celular, se creará un conjunto de datos a partir de la herramienta COBRApy que calcula el flujo metabólico celular de bacterias como *E. coli*.

La segunda propuesta es la integración de Algoritmos Genéticos con el fin de optimizar productos metabólicos a través de la mínima concentración de reactivos. Para ello se crea una población de individuos que representan los metabolitos y mediante el proceso de selección, reproducción y mutación se obtendrá el individuo mejor adaptado.

Por último, se presenta la metodología para integrar el modelo de Red Neuronal pre-entrenado al simulador GRO, aportando como trabajo futuro la aplicación de estas técnicas para evaluar modelos metabólicos de otras bacterias como Salmonella o el estudio de agentes celulares experimentales.

ABSTRACT

The technology is advancing constantly, and a clear example of this phenomenon is the development of Artificial Intelligence focused on the field of biology systems. Specifically, Neural Networks and Genetic Algorithms are highlighted as techniques applied to the search for patterns and the automation of mathematical calculations that could be inefficient by other techniques.

Important fields of the study and application in biological systems are the Agent-based models that are a type of models implemented by simulators that focus on modeling biochemical agents through reconstruction on a genomic scale of the metabolic network.

GRO is a synthetic multicellular simulator of genetic circuits based on agents that focus on the behavior of each individual in a colony by the metabolic model of each cellular agent. Due to the high parameters in a metabolic reaction on bacterial agents like, for instance *E. coli*, the computational cost and the time elapsed by executions, becomes inefficient to implement.

This research paper proposes to reduce computational cost by modeling cell metabolism in agent-based simulators. For this reason, it is proposed to analyze and evaluate Neural Network architectures for backpropagation. At the moment of writing this work, there is no training set based on cellular metabolism, so we have created a data set from the COBRApy tool that calculates the flux balance analysis through metabolic network in bacterium such as *E. coli*.

The second proposal is the integration of Genetic Algorithms in order to optimize metabolic products through the minimum concentration of reagents. For this purpose, a population of individuals representing the metabolites, is created and through the process of selection, reproduction and mutation the best adapted individual will be obtained.

Lastly, the methodology for integrating the pre-trained Neural Network model into the GRO simulator is presented and it contributes to the future work on the application of these techniques to evaluate metabolic models of other bacteria such as Salmonella or the study of experimental cellular agents.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad, el desarrollo tecnológico ha permitido descubrimientos y mejoras destinadas a comprender, estudiar y desarrollar el conocimiento en distintas áreas de la ciencia. Parte del actual desarrollo tecnológico ha sido gracias a la creciente investigación e implementación de técnicas en Inteligencia Artificial siendo pieza fundamental en el campo de la visión artificial, detección de patrones, reconocimiento de imágenes y rostros, así como un amplio desarrollo en la reducción de procesos matemáticos por automatización de funciones. Las Redes Neuronales y los Algoritmos Genéticos destacan en técnicas aplicadas al desarrollo de Inteligencia Artificial actual.

La biología de sistemas es un área multidisciplinaria con creciente desarrollo tecnológico y que comprende un campo de investigación en tecnología y biología enfocadas a estudiar, modelar y crear sistemas biológicos. La reconstrucción de redes bioquímicas es un enfoque de estudio de la biología de sistemas, siendo la reconstrucción de redes metabólicas una de las principales áreas de investigación. Hoy en día, existen líneas de investigación enfocadas en el Análisis del Balance de Flujo metabólico celular (FBA, por acrónimo en inglés) para calcular el flujo de metabolitos a través de una red metabólica, haciendo posible predecir la tasa de crecimiento o producción de bio-productos en un organismo.

Un enfoque de estudio de la biología de sistemas es la integración de simuladores computacionales con objeto de modelar sistemas biológicos a través de representaciones virtuales asistidas por computador, destinadas a la posterior implementación en laboratorios biológicos.

GRO es un simulador sintético multicelular de circuitos genéticos que cumple la función de emular el crecimiento poblacional de agentes bio-celulares. GRO ejecuta y emula las reacciones que ocurren durante el crecimiento celular con base al metabolismo de los agentes. Modelar sistemas basados en agentes microbianos encontrados en la literatura, implica la integración de sistemas con cientos e incluso miles de parámetros que integran el modelo metabólico asociado, haciendo qué la simulación requiera de un gran costo computacional. Sin embargo, los modelos metabólicos en GRO son creados por el usuario a partir de redes metabólicas sencillas y no pertenecientes a agentes microbianos reales.

En virtud de lo anterior, surge la motivación por aplicar técnicas de Inteligencia Artificial que aporten mejoras al costo computacional en simuladores de agentes microbianos como GRO. Mediante la aplicación de técnicas basadas en Redes Neuronales y Algoritmos Genéticos se pretende aportar herramientas en el estudio del crecimiento celular en ambientes restringidos.

Por tal motivo, la idea general de este trabajo de investigación es poder aplicar técnicas de Inteligencia Artificial enfocadas en el modelado de redes bioquímicas, promoviendo herramientas basadas en:

- Redes Neuronales capaces de predecir el Balance de Flujo metabólico, y
- Algoritmos Genéticos capaces de optimizar productos metabólicos a través de la mínima concentración de reactivos.

La presente investigación ha sido promovida como parte del PROGRAMA DE TITULACIÓN PARA EGRESADOS DE LA UNAM A TRAVÉS DE ESTANCIA ACADÉMICA EN EL EXTRANJERO (TEE) 2019 que convoca la Universidad Nacional Autónoma de México y desarrollada en las instalaciones de la Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid.

OBJETIVOS

Objetivo general

- Evaluar el empleo de Redes de Neuronas y Algoritmos Genéticos como métodos computacionales alternativos al modelado de redes metabólicas bioquímicas, de manera que aporten herramientas y mejoras en la predicción del Análisis de Balance de Flujo metabólico celular (FBA), siendo el marco de estudio la bacteria *E. coli*.

Objetivos específicos

- Estudiar los algoritmos de aprendizaje automático que resuelven problemas para datos lineales.

- Analizar las técnicas de redes neuronales de retro propagación para enfocarlas en la predicción eficiente del Balance de Flujo metabólico (FBA) en la bacteria *E. coli*.

- Comparar y analizar los resultados de precisión y tiempos de ejecución entre los diferentes modelos de Redes Neuronales en la predicción del Balance de Flujo metabólico (FBA) en la bacteria *E. coli*.

- Comparar y analizar los tiempos de ejecución de las redes neuronales artificiales y la ejecución de modelos de FBA por medio de la herramienta COBRApy.

- Aplicar metodologías en Algoritmos Genéticos capaces de optimizar la producción de productos metabólicos a través de la mínima concentración de reactivos, estudiando la bacteria *E. coli*.

1 MARCO TEÓRICO

Este capítulo se dedica a describir los conceptos esenciales relacionados con: Inteligencia Artificial, Biología de Microorganismos y Biología de Sistemas. Primero, se describen las ramas pertenecientes a la Inteligencia Artificial tales como: Aprendizaje Automático, Aprendizaje Profundo, Algoritmos Genéticos y Redes Neuronales. Con base en la implementación práctica de este trabajo, se presentarán los conceptos fundamentales en Biología de Microrganismos, Metabolismo Celular, Biología de Sistemas y Redes Metabólicas.

1.1 Inteligencia artificial

La inteligencia artificial o IA es una disciplina de la ciencia con gran relevancia en la actualidad, conceptualizada en el año 1956. Para sus inicios, la Inteligencia Artificial se encontraba limitada por falta de capacidad computacional en procesamiento de datos y la invención de algoritmos computacionales capaces de operar con grandes cantidades de información. Gracias al desarrollo de infraestructura computacional, en la actualidad se han podido realizar un sinfín de modelos basados en aprendizaje que buscan simular el comportamiento de agentes inteligentes como los humanos o los animales. Así, la Inteligencia Artificial ha permitido proveer herramientas para la comprensión del funcionamiento cerebral, detección de rostros, predicción de eventos, diagnósticos médicos, hasta proporcionar recomendaciones de vestimenta. Los evidentes resultados que han generado las investigaciones y prácticas en inteligencia artificial han permitido la gran expansión por la investigación en las diversas áreas del conocimiento humano.

1.2 ¿Qué es la inteligencia artificial?

La Inteligenciar Artificial es una rama de la ciencia destinada al estudio del comportamiento de agentes inteligentes conocidos hasta el momento por el ser humano, con el objetivo de poder crear máquinas capaces de: emular el comportamiento inteligente, aprender gracias a experiencias, contar con capacidad de adaptación al entorno para resolver tareas específicas, entre otras funciones.

Al ser una disciplina que engloba cualquier tipo de comportamientos inteligentes, se destaca un campo multidisciplinar presente en investigaciones de Matemática, Medicina, Física, Química, Tecnologías de Información, Biología, Economía, entre otras ramas de la ciencia.

El creciente estudio de la inteligencia artificial surge para el año 1956 ante la idea de entender la inteligencia natural humana y usar máquinas inteligentes que adquieran conocimientos y resuelvan

problemas considerados como intelectualmente difíciles. Ante esta idea, surgen dos enfoques de estudios en la inteligencia artificial:

-Desarrollo de modelos del comportamiento humano con aparatos cuya estructura se asemeje lo más posible a la del cerebro.
-Desarrollo de procesamiento con comportamientos inteligentes sin necesidad de presentar similitud al comportamiento humano.

Sin embargo, la principal problemática que se presenta en ambos enfoques es la capacidad para adaptarse a nuevas situaciones no evaluadas durante el proceso de desarrollo del modelo. Para resolver esta tarea se crean los *sistemas adaptativos*.

1.3 Sistemas Adaptativos

El desarrollo de sistemas capaces de adaptarse a nuevos entornos ha originado los *sistemas adaptativos* que funcionan en una autonomía de aprendizaje con base al conocimiento previo. Dependientes al entorno de estudio, los sistemas adaptativos cumplen con características específicas que constituyen las diferentes ramas de la Inteligencia Artificial.

1.3.1 Aprendizaje Máquina

El Aprendizaje Máquina, mejor conocido como Machine Learning, es una rama de la Inteligencia Artificial que busca cómo transmitir a las máquinas la capacidad de aprendizaje a partir de un conjunto de experiencias.

El Aprendizaje Máquina trata sobre cómo hacer que las máquinas modifiquen o adapten sus acciones (control, presión, etc.), por medio de una medida de precisión con respecto a la realidad. En problemas de clasificación, la precisión mide el nivel de confianza en que la entrada del sistema sea correspondida a cada etiqueta [1]. Los sistemas de Aprendizaje Automático buscan encontrar la forma de transmitir a las máquinas la capacidad de resolver tareas específicas, encontrar patrones y analizar datos con menor tiempo en el que una persona lo podría realizar.

1.3.2 ¿Cómo se analiza un problema usando el Aprendizaje Máquina?

Para entender la forma de analizar diversos problemas con Aprendizaje Máquina, a continuación se presenta la definición expuesta por el autor Tom Mitchell en 1997 sobre Machine Learning y posteriormente ejemplificarla con investigación realizada por Google en la detección de retinopatía diabética desarrollada en el 2016.

> *"Un programa de computadora es capaz de aprender desde una experiencia E con respecto a una tarea T y medido por un valor de rendimiento P, si su medida de rendimiento P se ve mejorada por la experiencia E." – Tom Mitchell, 1997*

En el 2016 se publicó el estudio de los trabajos que ha realizados Google en la detección de la retinopatía diabética mediante el análisis de imágenes con pacientes que presentan este padecimiento.

Con este ejemplo y por medio de la definición de Tom Mitchell se puede marcar la tarea T como "Paciente con alta probabilidad de retinopatía diabética". La experiencia E es regida por los datos de entrenamiento pertenecientes al conjunto de imágenes en pacientes con retinopatía diabética y pacientes considerados como sanos. La medida de rendimiento P puede ser definida como la efectividad de la clasificación o la medida de la certeza de precisión. Para ello, el estudio de caso se puede trabajar mediante las siguientes secciones:

A. Establecer las características en la obtención de una imagen de fondo de ojo, para este punto es necesario tomar en cuenta las consideraciones óptimas y la normalización[1] de los datos.
B. Desarrollar un algoritmo que detecte los patrones presentes entre las imágenes analizadas por medio de las etiquetas: "Paciente con alta probabilidad de retinopatía diabética" y "Pacientes sanos".
C. Fase de comprobación y pruebas de precisión en los casos analizados, repitiendo los pasos A y B hasta encontrar los resultados considerados como satisfactorios.

Para ello, los algoritmos de clasificación conllevan a una serie de técnicas de tratamiento de datos para la búsqueda de patrones presentes en cada etiqueta. Algoritmos más certeros precisan una cantidad mayor de datos y la constante actualización de éstos.

1.4 Tipos de aprendizaje

1.4.1 Aprendizaje Supervisado

En este tipo de aprendizaje, los datos de entrenamiento incluyen el nombre de la solución deseada, llamada etiqueta. Un claro ejemplo del aprendizaje supervisado son las tareas de clasificación. "En el aprendizaje supervisado es donde se aprende a través de los datos que han sido clasificados con la respuesta real" [2].

[1] La normalización de datos implica operar con la misma resolución para todas las imágenes, distancia focal, efectos de imagen, entre otras características.

Para el Aprendizaje Supervisado, un atributo se considera como un dato y las características son dependientes del contexto. Así pues, un algoritmo de clasificación también puede brindar valores numéricos referentes a una regresión. En el caso de ejemplo de la sección 1.3.2 podríamos establecer: una imagen analizada contiene una probabilidad del 30% de ser "Paciente con alta probabilidad de retinopatía diabética" y 70% de ser "Paciente sano".

Algunos de los algoritmos que integran el aprendizaje supervisado son:

-Regresión Lineal,
-Regresión Logística,
-Árboles de decisión,
-Redes Neuronales,
-Máquinas de Vector Soporte.

1.4.2 Aprendizaje No Supervisado

Aprendizaje No Supervisado implica que los datos de entrenamiento no han sido etiquetados previamente, por lo que el sistema intenta aprender sin un previo conocimiento de la clasificación de los datos. Algunos de los algoritmos presentes en el Aprendizaje No Supervisado son:

-Clustering,
-Visualización y reducción de dimensionalidad,
-Aprendizaje por Reglas de asociación.

Estos algoritmos suelen ser usados para clasificar grupos en una población donde no se tiene la certeza en su clasificación. Gracias al comportamiento, características o patrones, un algoritmo de aprendizaje no supervisado sí es capaz de clasificar.

1.4.3 Aprendizaje Reforzado

Se considera un aprendizaje entre el supervisado y no supervisado. Una vez dada una salida, se le indica al algoritmo que la respuesta está mal, sin señalar el proceso de corrección. Encontrar las mejores respuestas conlleva a un trabajo de prueba y error hasta llegar a una solución satisfactoria.

1.4.4 Aprendizaje Profundo

El aprendizaje profundo es una metodología basada en redes neuronales en las que su funcionamiento está inspirado en el sistema nervioso humano [3]. Con los estudios en neurociencia se ha podido estudiar el funcionamiento del sistema nervioso humano, capaz de crear redes neuronales artificiales que se enfoquen en tareas específicas. En el campo de la visión asistida por computadora, un grupo de neuronas es la encargada de detectar colores, líneas, bordes o incluso detectar rostros humanos.

Mediante el concepto de dividir una tarea entre grupos de neuronas artificiales que se especializan en tareas específicas, el aprendizaje profundo toma su forma de funcionamiento. Así, el aprendizaje profundo se puede definir como un modelo de redes neuronales que por su arquitectura es capaz de detectar patrones específicos presentes en un problema determinado [3].

Los métodos en aprendizaje automático pueden expresarse como un conjunto de capas de unidades de procesamiento que unidas trabajan para satisfacer un objetivo específico.
El autor Josh Patterson [4] define al aprendizaje profundo como un tipo de redes neuronales con un gran número de parámetros y que pueden clasificarse según sus características en:

- Redes no supervisadas pre-entrenadas,
- Redes convolucionales,
- Redes neuronales recurrentes,
- Redes neuronales recursivas.

1.5 Redes Neuronales Artificiales

1.5.1 Neurona Biológica

La neurona biológica es la célula que provee las funciones fundamentales del sistema nervioso de los seres humanos. La comunicación entre neuronas se hace capaz gracias a impulsos electroquímicos a través de la sinapsis entre neuronas. Para que una neurona pueda transmitir impulsos químicos a otras, esta debe de sobrepasar un umbral mínimo.
Como se puede ver en la Figura 1.1 la neurona se compone de un cuerpo celular llamado soma y compuesto de diversas dendritas, pero un único axón con ramificaciones. Las dendritas son estructuras delgadas que surgen del cuerpo celular y cuya función es entrar en contacto sináptico con otras neuronas. Los axones son los encargados de llevar la información del cuerpo de la neurona hasta la siguiente neurona. Se le denomina sinapsis a la conexión de un axón y las dendritas de otra neurona permitiendo también crear circuitos neuronales.

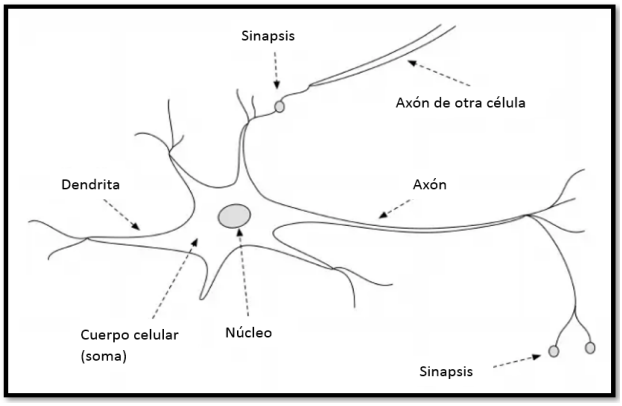


Figura 1.1 Neurona Biológica.
"The biological neuron". Figura tomada de: Josh Patterson, Adam Gibson, Deep Learning A Practitioner´s Approach, O'Reilly, agosto 2017

1.5.2 Neurona Artificial y Redes Neuronales Artificiales

La construcción de redes neuronales es a través de neuronas artificiales. Estas son las unidades simples, formadas por señales de entrada con asignación de pesos específicos produciendo señales de salida usando una función de activación. La neurona artificial es inspirada en la neurona del sistema nervioso humano. Como se observa en la Figura 1.2, se compone por una sección de vector de entrada x_i con sus respectivos pesos por entrada (dendritas en una neurona biológica, ver Figura *1.1*). Al interior de la neurona se encuentra una función de agregación (cuerpo celular, ver Figura *1.1*) y, por último, la salida de la neurona proporciona un valor a las neuronas conectadas (referente al axón a una neurona biológica, ver Figura 1.1*)*.

Las Redes Neuronales Artificiales (en adelante RNA) son un modelo computacional de procesamiento de datos que ha sido inspirado en el funcionamiento del sistema nervioso humano. La base del funcionamiento en las RNA resulta de intentar emular el comportamiento en red de interconexiones entre cada uno de sus elementos llamadas neuronas artificiales. Constan de elementos organizados en capas que han de emular la sinapsis neuronal.

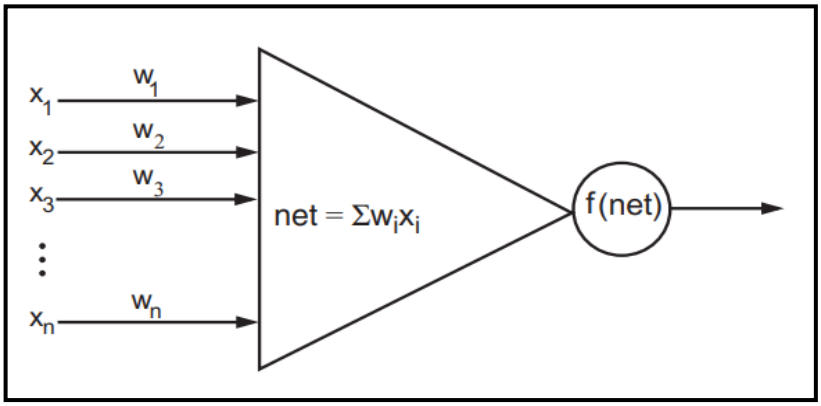


Figura 1.2 Representación ilustrativa de una neurona artificial. A la entrada de la neurona el vector x_i afectado por sus presos w_i, en el cuerpo la función de agregación $net = \sum x_i * w_i$ y la salida de la neurona $f(net) = \sum x_i * w_i$.
Luger, George F. Artificial intelligence: structures and strategies for complex problem solving / George F. Luger. 6th ed.

Algunos de los estudios importantes en RNA que cabe destacar son los siguientes:

1) En 1946 Donald Hebb propone un sistema de aprendizaje para la modificación de la sinapsis denominado “Regla de Aprendizaje Sináptico” o “Regla de Hebb”, que consiste en que una conexión entre neuronas se refuerza cada vez que es utilizada.

2) A finales de los 50, Widrow diseña el Adaline (Neurona Adaptativa Lineal) y Rosenblatt el Perceptrón: una máquina con un comportamiento adaptativo capaz de reconocer patrones, dotada de la regla del aprendizaje denominado de “auto asociación”, donde el estímulo y la unidad de respuesta están asociados por la acción de las entradas.

3) En 1969 Minsky y Papert publican "*Perceptrons: An Introduction to Computation Geometry*", donde realizan una seria crítica del Perceptrón, debido principalmente a su naturaleza lineal.

Principio de Funcionamiento

Una neurona recibe entradas y proporciona una o más salidas, gracias a diversas características:

Pesos

Corresponde a un valor numérico real que se asigna como medida de la contribución o importancia que aporta una entrada respecto a la salida de la neurona. Dicho valor puede ir cambiando a medida que el sistema se ajusta. Durante el principio del entrenamiento de la RNA, los pesos se determinan de forma aleatoria. Idealmente los valores de los pesos se encuentran entre 1 y -1.

Umbral

El umbral o sesgo, que permite indicar cuando debe o no reaccionar la neurona.

Bias

El *bias* es un valor escalar que se añade a la entrada para garantizar que al menos unos nodos de la capa se activen independientemente de la intensidad de la señal de entrada. Permiten a la red probar nuevas interpretaciones o comportamientos. El *bias* es representado por la letra *b* y al igual que los pesos, son modificados durante el proceso de entrenamiento de la RNA.

Aprendizaje

El aprendizaje va a consistir en encontrar para cada neurona de la red los mejores valores para obtener la salida esperada. Cuantas más entradas tenga una neurona, más pesos tendrá que ajustar y más completo será el aprendizaje [3].

Función de agregación

La Función de Agregación es expresada matemáticamente como una suma ponderada de las señales de entrada X multiplicadas por sus respectivos pesos W. Se puede decir que la función de agregación expresa la importancia que recae en cada una de las entradas de la neurona para que esta pueda transmitir o no una salida a las neuronas interconectadas.

La función de agregación puede expresarse de la siguiente manera:

$$\sum_{i=1}^{n} X_i * W_i \qquad Ecuación\ 1$$

Función de activación

La Función de Activación se expresa como una suma ponderada correspondiente a los pesos de las entradas de una neurona y dando como resultado un valor único como resultado de la suma.

Una vez calculado el valor único, la neurona compara este valor con un umbral y decide la salida. Por ejemplo, si el resultado de la suma se encuentra sobre un umbral de 0.5 la salida de la neurona será un 1.0, de lo contrario la salida será 0.0. Se puede representar matemáticamente de la siguiente manera:

$$O(x_1, \dots, x_n) = \begin{cases} 1 & si \quad w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0 \\ -1 & si \quad w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n < 0 \end{cases} \qquad Ecuación\ 2$$

Debido a la naturaleza de los problemas con los que se llega a trabajar en las RNA, existen diversas funciones de activación que cumplen propósitos específicos. A continuación, se presentan algunas de las más utilizadas:

• Función Heaviside

La Función Escalón Unitario, o Heaviside en inglés, es una función muy simple: devuelve +1 o 0. De este modo, si el valor de la función de agregación es mayor que el umbral, devuelve +1 en caso contrario devuelve -1 (o 0 según sea la aplicación).

Esta función es utilizada en problemas de clasificación para saber si un objeto es o no de una clase correcta. Con base en la Figura 1.3 A podríamos destinar dos posibles valores (1 y -1) para etiquetar 2 elementos de una clasificación. Puede también implementarse en otras aplicaciones, pero es difícil de usar, puesto que no indica en qué medida pertenece un valor a cada clasificación.

• Función Sigmoide

La Función Sigmoide se expresa matemáticamente como:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Está comprendida entre valores 0 y 1, con un valor igual a 0.5 en 0. En la neurona el método se invoca con f(x) = $valor\ agregado - umbral$. De este modo, se tiene una salida superior a 0.5 si el valor agregado es mayor que el umbral e inferior a 0.5 en caso contrario. La función permite un mejor aprendizaje gracias a su pendiente. Como se observa en la Figura 1.3 B, la salida de la función presentará un gran número de posibles valores. Además, permite comprender hacia dónde dirigirse sobre la pendiente de la función.

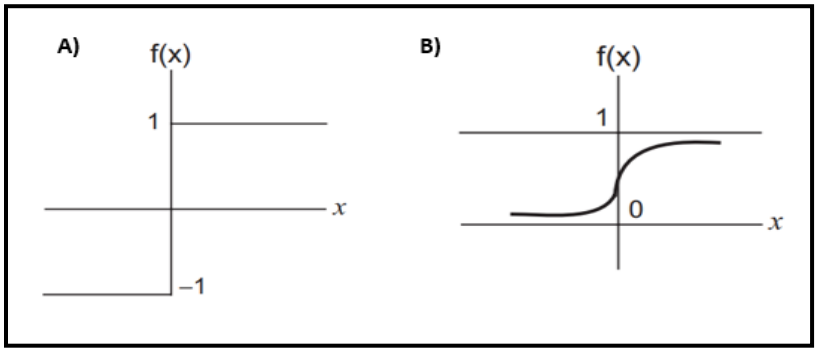


Figura 1.3 A) Representación de la función Heaviside B) Representación de la función Sigmoide

• Rectificación Lineal

La Rectificación Lineal (RELU) es una transformación que activa un nodo solo si la entrada está por encima de cierta cantidad. Mientras la entrada está por debajo de cero, la salida es cero, pero cuando la entrada sube un cierto umbral, tiene una relación lineal con la variable dependiente [4], demostrado en la Figura 1.4.

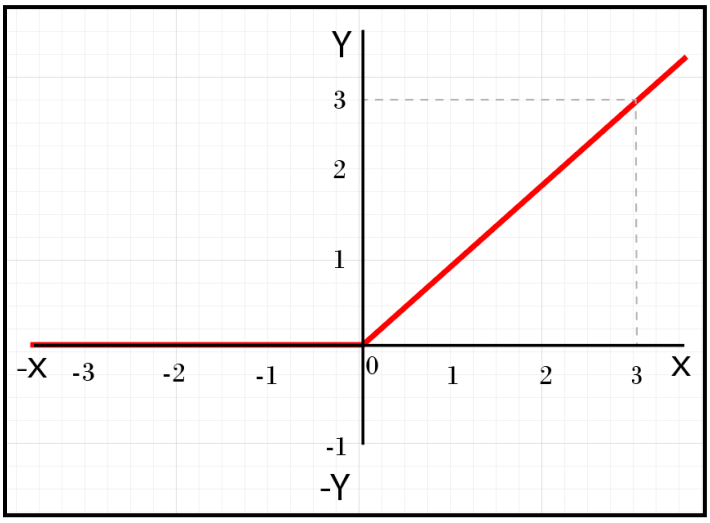


Figura 1.4 Función de activación: Rectificación Lineal (RELU)

• Función Softmax

La regresión logística multinomial (Softmax) es una generalización de la función sigmoide utilizada ampliamente en redes neuronales. Softmax brinda una distribución de probabilidad en las diferentes clases, siendo ampliamente utilizada en clasificaciones no binarias. Su expresión matemática se expresa:

donde $\boldsymbol{x}$ es un vector de entradas hacia la capa de salida

$$\sigma(z)_{\mathrm{j}} = \frac{e^{xj}}{\sum_i e^{xi}}$$

j = 1,2, 3, …, k. Hace referencia al índice de las salidas de clasificación

• Función ELU

Unidad Lineal Exponencial (ELU) mejora el gradiente de la función RELU al integrar valores negativos en su expresión como se representa en la Figura 1.5, impulsando la activación en los valores más cercanos a cero. Los autores Djork-Arné Clevert, Thomas Unterthiner y Sepp Hochreiter afirman que el aprendizaje es más rápido y generalizado significativamente respecto a la función RELU.

$$\boldsymbol{f(x)} = \begin{cases} \mathbf{x}, & \mathbf{x > 0} \\ \boldsymbol{\alpha}\,\mathbf{exp}(\boldsymbol{x}) - \mathbf{1}, & \boldsymbol{x} \leq \mathbf{0} \end{cases}$$ expresión ELU

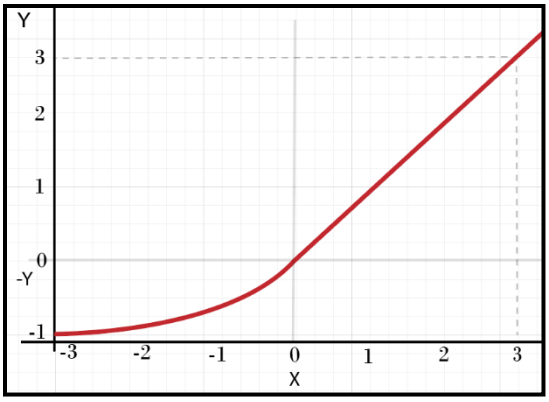


Figura 1.5 Función de activación: Unidad Lineal Exponencial (ELU).

• Función SELU

Unidad Lineal Exponencial Escalada (SELU) es una variante de ELU con la característica de converger en la media y varianza unitaria cuando es propagada en redes neuronales multicapa. La Figura 1.6 representa a la salida y cuando el valor de **x** es menor o igual a cero, la función se aproxima a cero y cuando **x** es mayor a cero, se multiplica **λ** por el valor de **x**. Su función se expresa:

$$f(x) = \lambda \begin{cases} x, & x > 0 \\ \alpha \exp(x) - \alpha, & x \leq 0 \end{cases}$$

Los valores de α es aproximado a 1.6733 y λ ≈ 1.0507

Figura 1.6 Función de activación: Unidad Lineal Exponencial (SELU).

1.5.3 Perceptrón

El Perceptrón fue creado por Frank Rosenblatt en 1957 y es un modelo unidireccional compuesto por dos capas de neuronas, una de entrada y otra de salida. Las neuronas de entrada son discretas y la función de activación de las neuronas de la capa de salida es de tipo escalón [5].

El Perceptrón Simple puede utilizarse como clasificador lineal gracias al algoritmo de aprendizaje introducido por Rosenblatt en 1962 que permite determinar automáticamente los pesos sinápticos. El algoritmo de aprendizaje del Perceptrón Simple ajusta los pesos de manera proporcional a la diferencia existente entre la salida actual de la red neuronal y la salida deseada, con el objetivo de minimizar el error actual de la red [6], donde el proceso de aprendizaje es iterativo. Sin embargo, los estudios realizados por Minsky y Seymour Papert demostraron que el Perceptrón no es capaz de resolver problemas linealmente separables.

El Perceptrón se representa como una función tipo Heaviside con un umbral de 0.5 arrojando un valor binario (0 o 1), dependiendo de la entrada. En la Figura 1.7 podemos ver la representación del Perceptrón Simple que añade el término bias w_0 . El *bias* es la capacidad de cambiar los límites de decisión del modelo, sin afectar los valores de entrada.

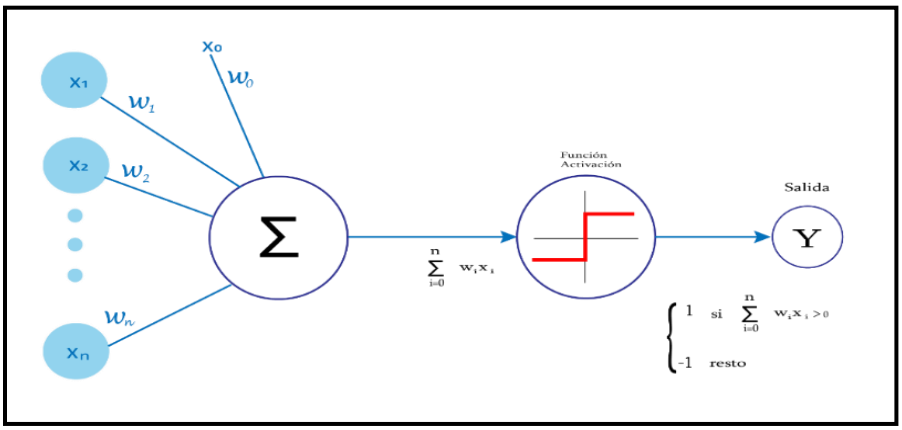


Figura 1.7 Representación del Perceptrón Simple.

1.5.4 Entrenamiento de Redes Neuronales Artificiales

El proceso de entrenamiento de una RNA permite calcular y asignar el nivel de importancia a cada entrada de una neurona artificial, permitiendo ajustar los pesos y el *bias* a medida que el valor de la salida se ajusta a la realidad. Las RNA aprenden mediante la relación que llega a existir entre la entrada-pesos y qué tan preciso es el resultado esperado.

Las funciones de pérdida en los algoritmos de optimización, como el descenso de gradiente estocástico (SGD), recompensan a la red por las buenas suposiciones y las califican por las malas. SGD mueve los parámetros de la red para hacer buenas predicciones y alejarse de las malas [4].

1.5.5 Descenso del Gradiente

El algoritmo del Descenso del Gradiente es un método implementado para optimización de funciones no-convexas y diferenciables en todo su dominio. Este algoritmo parte mediante la evaluación de la función de coste evaluada en un punto **P** escogido aleatoriamente o definido como

base de partida (solución conocida por expertos, ya que parte de un punto sin referencia y otorga mayor amplitud de búsqueda). Con el primer punto dado se calculan las derivadas parciales a cada parámetro de la función en dicho punto, dando como resultado el vector de dirección hacia donde la pendiente asciende. Así, para minimizar la función de coste bastaría con restar el vector gradiente a la función de evaluación. Una vez dada una solución, se calculan las soluciones vecinas, a una distancia de una unidad llamado *ratio de aprendizaje*. Este valor indica qué tanto afecta el gradiente a la actualización de los parámetros entre cada iteración.

1.5.6 Arquitectura de Red Neuronal

La arquitectura de una Red Neuronal se compone por el conjunto de neuronas agrupadas en capas, donde cada capa cumple una función específica según su posición en la Red Neuronal, de la siguiente manera:

Capa de entrada: Las neuronas de la capa de entrada reciben los datos que se proporcionan a la RNA para que los procese.

Capas ocultas: Se les conoce como todas aquellas capas que sus conexiones no se encuentran expuestas a los datos de entrada o salida de la red. Estas capas introducen grados de libertad adicionales en la RNA. El número de ellas puede depender del tipo de red que estemos considerando y el problema a tratar, además de ser las encargadas de gran parte del procesamiento.

Capa de salida: Esta capa proporciona la respuesta de la Red Neuronal. Normalmente también realiza parte del procesamiento.

1.6 Arquitecturas de entrenamiento

1.6.1 Perceptrón Multicapa

Consiste en un tipo de arquitectura de red neuronal comprendida por una capa de entrada, una o más capas ocultas, y una capa de salida. Se caracteriza por que todas las capas se encuentran unidas por conexiones hacia adelante, pero sin enlaces entre neuronas artificiales de la misma capa o enlaces retroalimentados entre capas. Cada neurona se encarga de propagar la información hacia las neuronas de la siguiente capa mediante su función de activación correspondiente.

La Figura 1.8.1 representa el diagrama de una neurona artificial que pertenece a una Red Perceptrón Multicapa que a diferencia de la Figura 1.5 se puede observar una función de activación y una salida con valor numérico y no booleano como la presente en el perceptrón simple. La entrada de la función de activación sigue siendo la suma de los productos de la entrada por los pesos, pero la salida contribuye a un número mayor de posibles resultados.

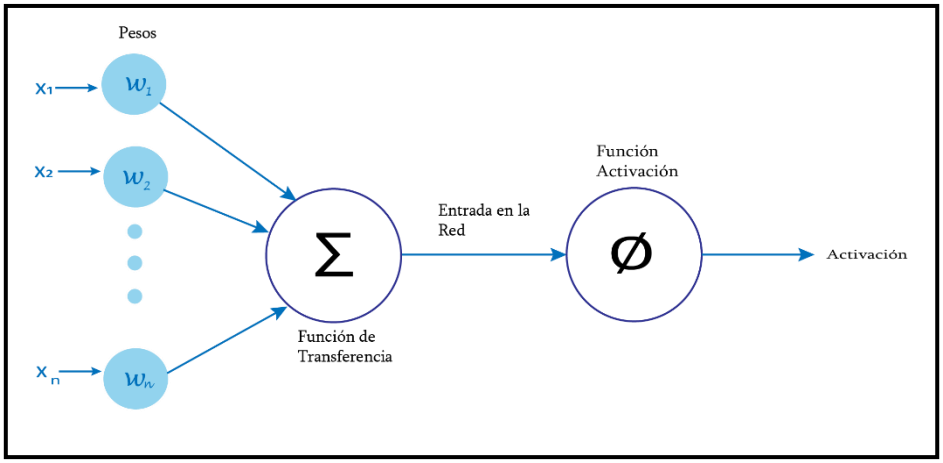


Figura 1.8.1 Neurona Artificial de un Perceptrón Multicapa
Josh Patterson, 2017.

Las capas de entrada tienen la función de tomar los valores específicos del índice del vector de entrada y la función de activación se considera lineal; sin embargo, en las capas ocultas las entradas de una neurona son el resultado de la función de activación de otras neuronas previas a ésta.
Se puede expresar la entrada de la red con *bias b* de la siguiente manera:

$$Vector\ de\ entrada_{neurona}\ \ I\ =\ W_i \cdot A_i + b$$
$$Salida_{neurona}\ \ a_i\ = g(\ W_i \cdot A_i + b)$$

Donde W_i es el vector de todos los pesos entrantes en la neurona i y A_i es el vector de los valores de activación para las entradas de la neurona.
Los datos de entrada son los encargados de producir información, los pesos y *bias* son los encargaos de cuantificar la actividad de la neurona dentro de la RNA.

1.6.2 Algoritmo de Aprendizaje por Retropropagación

El algoritmo de Retropropagación (Backpropagation) es sumamente utilizado para reducir el error durante el proceso de entrenamiento de una RNA. Gracias a su forma de operación, permite asignar el nivel de importancia o responsabilidad de cada neurona artificial en una RNA. Para describir cómo funciona el algoritmo de Backpropagation se utilizará un ejemplo práctico y descrito por medio en las Figura 1.8.2 y Figura 1.8.3.

Para que un producto pueda ser consumido por nosotros previamente ha tenido que pasar por diversas etapas. En el caso de la entrega de comida a domicilio, las etapas de elaboración pueden incluir: cultivo de ingredientes, compra de ingredientes, elaboración y transporte para la entrega. La medida de reputación de un comercio puede ser medido por la satisfacción o recomendación que los usuarios dan al producto en una plataforma. Una forma para mejorar la confianza del

cliente es analizando el nivel de responsabilidad que tiene cada etapa y qué tan importante resulta al final. Analizando el error desde la entrada, como se muestra en la Figura 1.8.2, debemos evaluar la responsabilidad desde la capa inicial hasta la capa de salida; sin embargo, no sería posible determinar en qué medida influyen las capas intermedias respecto a la salida.

El algoritmo de Backpropagation se encarga de evaluar la importancia de cada etapa respecto a la señal de error propagando el error hacia atrás. Encontrando una neurona que tenga mayor influencia en el error, se le responsabilizará con parte de ese error. Con el ejemplo anterior, podemos analizar si el error depende principalmente del servicio de entrega a domicilio o si por el contrario depende más de la elaboración del producto. Es decir, si observamos satisfacción del cliente, no depende del servicio de transporte. Entonces, tampoco afectará a las capas previas (elaboración, compra de ingredientes y cultivo de ingredientes). Si, por el contrario, gran parte del error recae en la capa de compra de ingredientes, entonces el algoritmo responsabilizará a dicha capa. De esta forma, es posible establecer el nivel de responsabilidad de cada capa respecto al error.

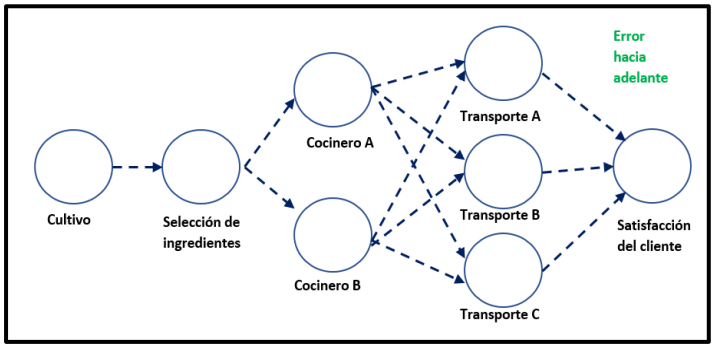


Figura 1.8.2 Representación del Algoritmo de Propagación hacia adelante o Feedforward.

La propagación hacia atrás provee un algoritmo que reparte el problema y ajusta los pesos con el objetivo de minimizar el error. El enfoque de la propagación hacia atrás comienza con la capa de salida y propaga el error sobre las capas ocultas. Otro interesante aspecto de las redes multicapa es el papel que juegan las capas ocultas. Las capas ocultas permiten generalizar la red neuronal. Esto quiere decir que menos nodos en las capas ocultas son usados para codificar los patrones de la información en el proceso de entrenamiento.

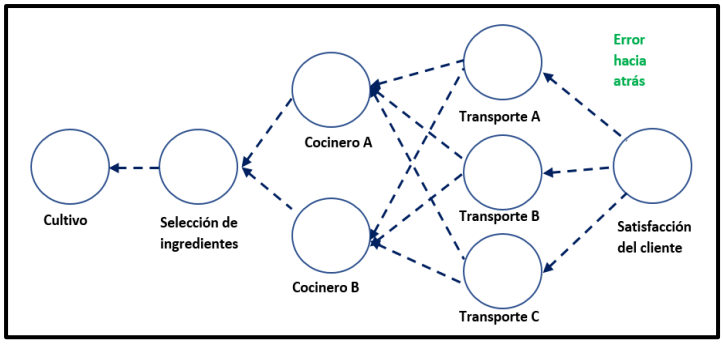


Figura 1.8.3 Representación del Algoritmo de Propagación hacia atrás o Backpropagation

1.7 Algoritmos Genéticos

1.7.1 Conceptos de Algoritmos Genéticos

Los Algoritmos Genéticos son considerados dentro del conjunto de los algoritmos evolutivos. La esencia de estos Algoritmos Genéticos es tomar una población de individuos y aplicar la selección natural. El algoritmo inicialmente toma un grupo al azar de la población y mediante el procesamiento de los datos elige el individuo más fuerte entre ellos. La selección del individuo más fuerte es efectuada mediante una función de evaluación o también llamada función de *fitness*.

Gracias a la selección de los individuos es posible crear una nueva generación por recombinación y mutación (también suele llamarse herencia por padres). En el caso de los Algoritmos Genéticos, los individuos son tratados como una cadena de valores, donde cada valor es una codificación de una posible solución. Por lo tanto, la función de evaluación determina qué tan bien una cadena de datos puede ser considerada apta para resolver el problema [7]. Es importante señalar que estos algoritmos siguen los principios de procesos no-deterministas por lo que al correr el algoritmo en múltiples escenarios corresponde a tener múltiples soluciones.

Como se ha mencionado, los Algoritmos Genéticos parten de los principios de la selección natural de individuos en una población. Es por ello que encontramos terminologías biológicas cuando hablamos sobre la metodología que sigue el desarrollo de estos algoritmos. Parte de estas terminologías se presentan a continuación:

Gen: Representa la herencia de un carácter. En Algoritmos Genéticos designa una posible solución a un problema por medio de unos parámetros denominados *genes* que son codificados en una cadena de vectores llamada cromosoma.

Tamaño de población: Parámetro que indica el número de cromosomas en una población como parte de una generación. El tamaño de población es un factor importante para que el algoritmo pueda encontrar soluciones óptimas en un mayor o menor tiempo. Un Algoritmo Genético realiza cambios aleatorios en uno o más individuos de la generación actual para producir una nueva solución candidata. El proceso de selección se lleva a cabo utilizando una función de condición física que calcula la fuerza de cada individuo.

Genotipo: El conjunto de los parámetros representado por un cromosoma particular recibe el nombre de genotipo. El genotipo contiene la información necesaria para la construcción del organismo, es decir, la solución real al problema, denominada *fenotipo*.

Fenotipo: Características físicas de un organismo, atribuibles a la expresión de su fenotipo. Contiene tanto los rasgos físicos como los conductuales. Es el resultado de la interacción entre el genotipo y el ambiente; se interpreta como la suma de los caracteres observables en un individuo.

Cromosoma: Es la molécula única de ADN, unida a histonas (proteínas básicas) y otras proteínas que se condensa durante la mitosis (proceso de división celular — reparto equitativo del material hereditario) y la meiosis (proceso de fragmentación — divisiones pequeñas), formando una estructura compacta. Un cromosoma en Algoritmos Genéticos se representa por una cadena de bits.

Alelo: Representa el valor de un gen. Una de las dos o más formas alternativas de un gen, determina el carácter controlado por éste. Al representarse en forma de bits toman valores de 0 y 1.

Función de aptitud: Es un tipo especial de función que cuantifica que tan óptima es una función. Se traduce en un cromosoma óptimo para que sus bases sean combinadas con cualquier otra técnica para la producción de una nueva generación que sea mejor a las anteriores.

Selección: Metodología por la cual se seleccionan los cromosomas para su reproducción de acuerdo con la función de aptitud. Para su aplicación se utiliza una distribución de probabilidad, donde la probabilidad de selección de una cadena es directamente proporcional a su aptitud.

Cruzamiento: Método de fusión sobre la información genética de dos individuos con la función de proveer descendientes sanos. Es el encargado de heredar información. En este proceso se intercambian genes entre cromosomas de los dos progenitores. La metodología de *cruzamiento por único punto* que se muestra en la Figura 1.9, consiste en seleccionar un conjunto de la población de entre los progenitores. Con dicha selección se realiza la cruza para conformar los descendientes de la nueva generación, obteniendo dos individuos diferentes como se observa en la Figura 1.9.

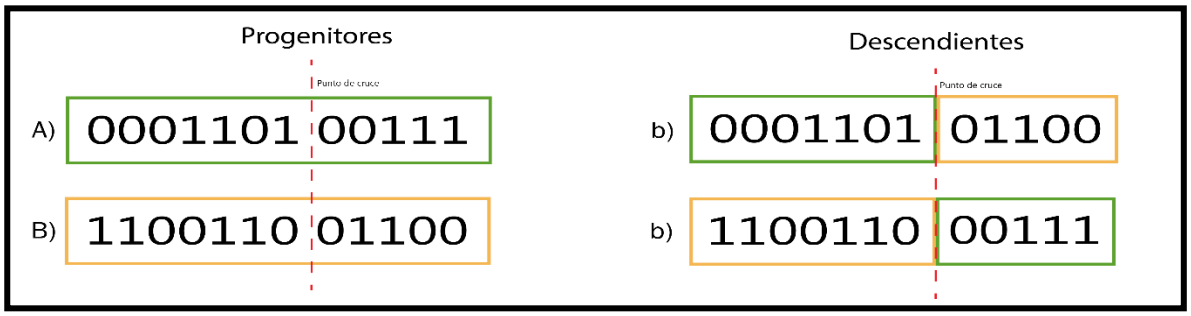


Figura 1.9 Cruzamiento de único-punto.
Pedro Ponce Cruz

Mutación: La mutación consiste en la inversión del gen mutado que corresponderá con un bit. Produce cambios incrementales al azar en la descendencia, efectuando cambios aleatorios en los valores de los alelos. Es un proceso importante ya que brinda una manera confiable de introducir nuevas características que no se encuentran presentes en los progenitores.

1.7.2 Metodología del Algoritmo Genético

Los Algoritmos Genéticos utilizan una metodología iterativa para encontrar soluciones óptimas a partir de la evaluación de la función de aptitud por cada población generada. La función de aptitud debe de considerarse eficientemente para que así produzca una solución coherente al problema dado, de lo contrario existe la posibilidad de encontrar soluciones falsas o aquellas que no satisface al sistema. Al Implementar un Algoritmo Genético se requiere de una metodología iterativa:

- Codificar los datos de forma binaria,
- Generar una población inicial,
- Evaluación de la función fitness,
- Selección de una nueva población,
- Repetir los pasos de evolución y selección hasta elegir la de mejor puntuación.

En el capítulo 3 de esta investigación se llevará a cabo esta metodología para implementar algoritmos genéticos y aplicados al desarrollo práctico de la investigación para optimizar la simulación del Balance de Flujo Metabólico celular.

1.8 Conceptos de Biología

El desarrollo práctico de esta investigación se enfoca principalmente en Inteligencia Artificial aplicada a la simulación de sistemas biológicos. Estudiar los conceptos relacionados con bioquímica y Balance de Flujo Metabólico (FBA), permitirá comprender la terminología relacionada con la problemática de esta investigación y así mismo, involucrarse en el ámbito práctico del desarrollo de esta investigación abordada en el capítulo 3 de este trabajo.

1.8.1 Biología de sistemas

La Biología de Sistemas tiene la finalidad de diseñar y construir nuevas partes, dispositivos, sistemas y el rediseño de sistemas biológicos existentes con fines útiles [8]. La investigación e innovación ha creado nuevas tecnologías capaces de resolver problemas multidisciplinarios basados en la arquitectura molecular dando como resultado avances médicos como la detección y eliminación de tumores [22].

1.8.2 Metabolismo Celular

Todas las reacciones químicas celulares destinadas a la producción de energía se denominan metabolismo celular. Una serie de reacciones químicas donde el producto de una reacción es el sustrato para la próxima reacción se denomina vía metabólica. Las vías metabólicas incluyen dos tipos amplios: vías catabólicas y vías anabólicas.

Vías catabólicas: Es la parte del metabolismo que incluye los procesos relacionados con la degradación de sustancias complejas a sustancias más simples.

Vías anabólicas: Consiste en utilizar energía para construcción y agrupación de células para construir moléculas más grandes.

Estas reacciones ocurren en secuencia y no aisladamente, de manera tal que el producto de una reacción se convierte en el sustrato de la siguiente. Los organismos pueden clasificarse sobre la base de la identidad del agente oxidante utilizado para la degradación de los nutrientes. Los aerobios obligados deben utilizar O_2, mientras que los anaerobios facultativos, como *E. coli*, pueden crecer en presencia o en ausencia de O_2 [9].

Las vías metabólicas son una serie de reacciones enzimáticas conectadas que generan sus productos específicos. Sus reactivos intermediarios y productos se conocen como *metabolitos*. Los tipos de enzimas y metabolitos en una célula determinada varían con la identidad del organismo, del tipo de célula, estado nutricional y su etapa de desarrollo.

Las enzimas son moléculas orgánicas que actúan como catalizadores de reacciones químicas, acelerando la velocidad de reacción. Se puede decir que son proteínas que catalizan las reacciones químicas. En ausencia de las enzimas, las reacciones se producirían en forma tan lenta que no podrían sustentar la vida.

Un criterio importante al estudiar una vía metabólica es la identificación de las reacciones más importantes y las enzimas que las catalizan y que están sujetas a regulación por algún metabolito. Aunque el metabolismo celular involucra centenares de enzimas diferentes que catalizan las diversas reacciones, solo algunas de ellas regulan las vías metabólicas centrales [10].

1.8.3 Microbiota Humana

Es la colección de microorganismos que habitan el cuerpo humano y que se encuentra ligado con la salud y procesos cruciales del huésped en que habita [11]. La microbiota humana está presente en todos los seres humanos y se encuentra distribuida en diferentes partes del cuerpo como la piel, intestinos, entre otras partes del cuerpo humano. Contiene más de 100 billones de microorganismos incluyendo las bacterias y en su caso la microbiota intestinal ayuda a digerir algunos alimentos que el intestino delgado no es capaz de digerir.

La basta presencia de microorganismos en los seres vivos contribuye gran parte en la responsabilidad de la regulación metabólica del huésped [23]. Además, la microbiota humana tiene gran influencia en diversos agentes patógenos gastrointestinales, enfermedades circulatorias, la obesidad, además de tener gran importancia en la respuesta del sistema inmunológico ante la presencia de fármacos en el organismo huésped.

El estudio del metabolismo microbiano presente en la microbiota permite la comprensión de los cambios metabólicos específicos en un sistema complejo [11]. La realización de un estudio del metabolismo de agentes microbianos presentes en la microbiota humana, proporciona la base para una mejor comprensión del eje metabólico microbiano-humano y por lo tanto su implicación en la salud.

1.8.4 Redes Metabólicas

Las redes metabólicas nacen desde el concepto de estudiar el metabolismo celular como un conjunto de rutas metabólicas que existen en una célula y no como un aspecto reduccionista. Las vías metabólicas forman y funcionan como una red a través de la cuales los distintos metabolitos reaccionan de manera sucesiva hasta transformarse a componentes de biomasa. También se les suele denominar así al conjunto de interconexiones formadas por metabolitos en un organismo.

En el Anexo A, se presenta el modelo realizado por los autores Kim, H. & Lee, S. Y. [12] sobre la representación de la matriz estequiométrica basada en el modelo de red metabólica. En el capítulo 3 de este trabajo se detallará dicha representación.

1.8.5 Análisis de Balance de Flujo Metabólico

Como se explicó en la Sección 1.8.2, las vías metabólicas especifican las reacciones químicas involucradas durante el metabolismo celular. A la velocidad con la que se desarrollan las vías metabólicas se le conoce como flujo metabólico. El flujo metabólico involucra las concentraciones de los metabolitos intracelulares que describen la fisiología y regulación de nutrientes de cada célula. Mediante la determinación de los modelos de reacciones estequiométricas es posible estimar el equilibrio de masa para los metabolitos implicados durante el metabolismo celular [21].

El análisis del Balance de Flujo Metabólico (término conocido en inglés como Flux-Balance Análisis) es una técnica analítica que cuantifica los flujos metabólicos intracelulares y disecciona los aspectos funcionales de la red metabólica, buscando el desarrollo de modelos estequiométricos que puedan describir el metabolismo celular.

El análisis de flujo basado en restricciones es un término para realizar una optimización de técnicas de simulación. Primero, se construye un modelo estequiométrico basado en la información genómica encontrada en la literatura. Después es simulado por optimización lineal utilizando funciones objetivo (maximización de tasa de crecimiento celular) y construyendo las restricciones en espacio de solución por la capacidad celular [12].

La importancia del análisis de Balance de Flujo Metabólico recae en la representación matemática (matrices y vectores relacionan las reacciones estequiométricas) de los mecanismos que regulan el

comportamiento celular, siendo posible su análisis de las implicaciones y efectos que conllevan a la adición o eliminación de nutrientes relacionados con el metabolismo celular [13].

Es importante destacar que permite predecir y evaluar sistemáticamente los efectos de las perturbaciones genéticas y/o ambientales en la célula a escala global, y sugiere qué partes del sistema (por ejemplo, los genes) deben modificarse para la mejora directa del sistema [14].

Los modelos estequiométricos de varios microorganismos, incluida la *Escherichia coli*, con mucho el organismo mejor caracterizado, se han reconstruido, pero se limitaron a vías metabólicas principalmente centrales hasta mediados de los años noventa. En el caso de *E. coli*, su modelo estequiométrico se ha expandido gradualmente hacia una escala genómica al incorporar más información genética y bioquímica [16].

Con ayuda de la biología de sistemas es posible crear algoritmos capaces de representar las reacciones biológicas presentes en las células. Por lo que, con el estudio del análisis de Balance de Flujo Metabólico se parte la hipótesis de este trabajo en "*poder representar el metabolismo celular por medio de algoritmos de inteligencia artificial, con el fin de analizar los parámetros implicados en la regulación del comportamiento celular, así como predecir los efectos implicados en añadir nuevos nutrientes a una célula"*. De esta forma se aportarán nuevas investigaciones que ayuden a la simulación computacional de reacciones bioquímicas que de forma *in vivo* resultarían muy costosas.

En el siguiente capítulo se presentará el estado del arte referente al análisis del Balance de Flujo Metabólico, las herramientas computacionales en materia de Inteligencia Artificial, aprendizaje profundo para comportamientos lineales, así como mencionar los simuladores de microrganismos donde tendrá su implementación el desarrollo de este trabajo de tesis.

2 ESTADO DEL ARTE

En este capítulo 2 se describen los puntos esenciales dentro del estado del arte de este trabajo de investigación. Está dedicado principalmente en describir las herramientas y plataformas computacionales en materia de Biología de Sistemas para emular el Balance de Flujo Metabólico (FBA) de la bacteria *Escherichia coli*. De esta forma permitirá conocer los principios para desarrollar técnicas de I.A en la predicción del FBA y su aplicación en simuladores de bacterias.

2.1 Plataformas destinadas al desarrollo de Inteligencia Artificial

2.1.1 TensorFlow

TensorFlow es una librería de programación de alto nivel para entornos de desarrollo basados en Python, C++, Java, entre otros. La empresa Google provee el desarrollo y distribución de TensorFlow desde el 2005. TensorFlow posee un gran número de herramientas de cálculo numérico en vectores multidimensionales llamados tensores. La eficiencia de sus herramientas de cálculo numérico hace posible el desarrollo de Aprendizaje Profundo con TensorFlow.

Trabajar con Aprendizaje Profundo en TensorFlow ofrece un ambiente de trabajo óptimo para la implementación de Redes Neuronales Artificiales gracias a la integración de APIs optimizadas en rendimiento con GPU. Además, posee un ambiente de visualización en tiempo real que, entre otras cosas, permite observar el comportamiento de parámetros para modelos de arquitecturas de Redes Neuronales Artificiales, conocido como TensorBoard.

La integración de TensorFlow en este trabajo de investigación se especializará en la implementación de modelos de RNA en las plataformas de desarrollo en Python. Gracias a la comunidad de desarrolladores en TensorFlow, permite el análisis de modelos pre-entrenados que sirven como apoyo de investigación y comprensión del estado del arte en sistemas predictores de funciones matemáticas, RNA y biología de sistemas.

La ejecución de TensorFlow será llevada a cabo mediante el ambiente de desarrollo conocido como Jupyter Notebook[2]. Este último es un ambiente interactivo web que trabaja con código Python permitiendo crear modelos para aplicaciones de Machine Learninig, visualización de datos, y mucho más.

[2] Documentación y descarga de herramienta se encuentra disponible en: https://jupyter.org/

2.1.2 Keras

Keras es una Interfaz de Programación de Aplicaciones que integra TensorFlow para el desarrollo de Redes Neuronales Artificiales. Posee un alto soporte de herramientas para RNA capaz de modular de forma fácil arquitecturas de Redes Neuronales Recurrentes, Convolucionales, Backpropagation, entre otras. Al momento de realizar la investigación se encuentran herramientas de optimización, gráficos computacionales; especialmente en el entrenamiento y pruebas de modelos.

En este trabajo se utilizará Keras como motor de desarrollo en RNA, con el objetivo analizar nuestra hipótesis en modelar el Balance de Flujo Metabólico por medio de RNA. En los posteriores capítulos se implementarán los códigos de programación para la construcción del modelo de RNA, explicando la función que realiza cada línea y su contribución al desarrollo de este trabajo.

2.2 *Escherichia coli*, metabolismo y modelo matemático

Escherichia coli es una especie de bacteria que forma parte de la familia Enterobacteriaceae, descubierta en 1885 por Theodor Escherich. Desde su descubrimiento ha sido objeto de grandes estudios debido a sus capacidades patogénicas, diversidad poblacional y el estudio de su metabolismo.

La especie *Escherichia coli* K-12 MG1655 se considera la bacteria con mayor número de investigaciones, además de los diversos modelados metabólicos [16]. *E. coli* cuenta con un modelo representativo de red metabólica desarrollada en sus inicios por Edwars y Palsson en el año 2000. Investigaciones en su escala genómica han podido integrar al modelo nuevos genes, componentes de reacciones con proteínas, reacciones metabólicas y propiedades termodinámicas implicadas en los límites de concentraciones. La reconstrucción genómica para *E. coli* conocida como iAF1260 cuenta con 2077 reacciones, 1039 metabolitos y 1260 genes y ha sido utilizado para investigaciones y desarrollo de modelos predictivos.

El balance de flujo metabólico (BFM) puede ser usado como un modelo basado en restricciones para predecir la distribución de flujo, tasas de crecimiento y producción de biomasa. El modelo iAF1260 se mostró como una actualización a los modelos anteriores *(*K-12 MG1655, iJO1366*)* en *E. coli* implementando técnicas de BFM en la predicción de fenotipos y tasas de crecimiento, haciendo evidentes los estudios en el análisis de Balance de Flujo Metabólico celular [17].

2.3 Predicción del Análisis de Balance de Flujo [FBA] metabólico por herramienta computacional

En esta sección se presenta la herramienta COBRApy que ha sido desarrollada específicamente para el estudio del Balance de Flujo Metabólico en microorganismos. Con esto se muestran las características más importantes de esta herramienta y su importancia durante el desarrollo del trabajo de investigación.

2.3.1 COBRApy

El modelado basado en restricciones ha tenido grandes aplicaciones en el campo de la ingeniería metabólica microbiana [15]. El marco de estudio presente en los modelados basados en restricciones provee la facilidad de análisis en sistemas biológicos que pueden ser expresados por medio de parámetros. Las restricciones suelen estudiar parámetros en conservación de masa, crecimiento molecular, reacciones termodinámicas, así como la representación de modelos estequiométricos.

COBRApy es una librería desarrollada en Python que ha sido desarrollada con el propósito de modelar expresiones genéticas en escalas genómicas, presentando una herramienta para modelar el metabolismo celular con restricciones. El núcleo de COBRApy conserva el desarrollo del software COBRA Toolbox que corre en las distribuciones de MATLAB, incluyendo una compatibilidad y soporte de ambas plataformas. COBRApy comprende una programación orientada a objetos, con esquema de modelado e interacción de parámetros con soporte de comunidad[3].

Núcleo de COBRApy

El núcleo de las clases COBRApy son el Modelo, Metabolito, Reacción, y Gen. La clase Model es un contenedor de reacciones químicas, metabolitos asociados y productos genéticos. Dentro de un Modelo los metabolitos pueden ser modificados uno por uno o mediante la descripción de reacciones. Los parámetros Modelo, Reacciones, Metabolitos y genes, se encuentran referenciados entre ellos, por lo que mediante la llamada a una función es posible determinar los parámetros de otra clase. La programación orientada a objetos con COBRApy, permite acceder a los atributos de cada clase. Es posible acceder a la representación química de un metabolito, accediendo al atributo de la clase Metabolite.

Modelos Pre-Integrados

COBRAPy integra modelos de redes metabólicas para Salmonella entérica Typhimurium (modelo metabólico de los investigadores Thiele I, Hyduke DR y Steeb B) y *Escherichia coli* K-12 MG1655 [14]. Estos modelos pueden ser cargados mediante las funciones cobra.test.create_test_model. En el capítulo 4 de este trabajo se describirán las funciones utilizadas para cargar modelos, editar parámetros (metabolitos y reacciones), visualizar reacciones, entre otras funciones más.

El análisis del crecimiento en un medio determinado es un aspecto que busca estudiar el modelado basado en restricciones. Para ello, es necesario tener presentes los límites de concentraciones mínimas para un crecimiento, después realizar una optimización lineal para calcular el flujo máximo de concentración de biomasa [15].

[3] La comunidad y soporte de código abierto perteneciente a COBRApy se encuentra en: http://opencobra.sourceforge.net

COBRApy cuenta con funciones específicas para realizar el cálculo del Balance de Flujo metabólico mediante la representación de las ecuaciones estequiométricas de un metabolismo implicado. Estas funciones son capaces de calcular los valores de las concentraciones y crecimiento de biomasa respecto a los límites de producción de biomasa en ciertas condiciones.

El estudio de las concentraciones de metabolitos y la producción de biomasa en un ambiente de crecimiento establecido nos permitirá analizar y comprender qué parámetros están impidiendo o fortaleciendo el crecimiento de una bacteria.

2.4 Simuladores de microorganismos bacterianos

Los circuitos sintéticos conforman uno de los más recientes estudios de la biología sintética que se inspira en conocimiento del funcionamiento celular con la representación de circuitos electrónicos. Los sistemas biológicos basados en agentes es una metodología usada para representar el comportamiento celular en circuitos sintéticos, haciendo posible el estudio de individuos en una población. En este contexto, es posible representar a una célula como un individuo, respetando sus características independientes y su interacción con el entorno. De esta manera, se ha hecho posible el desarrollo de simuladores basados en agentes que ofrecen características de autonomía a cada individuo, interacción entre agentes o capacidad social (las características de un individuo pueden beneficiar o perjudicar a otros), dependientes del entorno y pueden ser dependientes únicamente de los límites y reglas de comportamiento [18].

Los modelos basados en agentes (ABM) son utilizados para simular el comportamiento de individuos cuando estos son dependientes en su totalidad del ambiente de desarrollo, sabiendo que cada uno puede tener características distintas. En el caso de la biología, el modelado de ABM se basa en la experimentación *in vivo*, conocimiento y literatura de los individuos a emular, por lo que debe haber un experimento real previo al desarrollo de las simulaciones para garantizar que los resultados obtenidos se acerquen a la realidad.

2.4.1 GRO

GRO (growing) es un simulador sintético multicelular de circuitos genéticos. Ha sido desarrollado por el Laboratorio de Inteligencia Artificial de la Universidad Politécnica de Madrid, siendo el lugar de investigación de este trabajo de tesis. Es un simulador basado en agentes, capaz de representar colonias de bacterias mediante agentes celulares independientes. La Figura 2.1 muestra el entorno de desarrollo de GRO. Su funcionamiento se basa en las reacciones que ocurren durante el crecimiento de biomasa de cada una de las células puestas en simulación.

GRO integra un conjunto de herramientas para simular señales básicas de interacción entre individuos, siendo capaz de realizar simulaciones gráficas de 105 bacterias en minutos [19]. CellPro es una herramienta integrada en GRO que simula la dinámica de expresiones genéticas mediante el uso de cálculos de concentraciones de proteínas sintéticas dentro del simulador. Esto

se hace posible gracias a la interpretación de presencia de proteínas por medio de valores lógicos binarios, haciendo más eficiente la representación de circuitos biológicos sintéticos [20].

La configuración metabólica de una célula en GRO es descrita por el usuario, por lo que logra la escalabilidad de simulaciones de microorganismos considerados como complejos. Con este principio se hace posible la idea de poder integrar modelos de metabolismos bacterianos como *E. coli*.

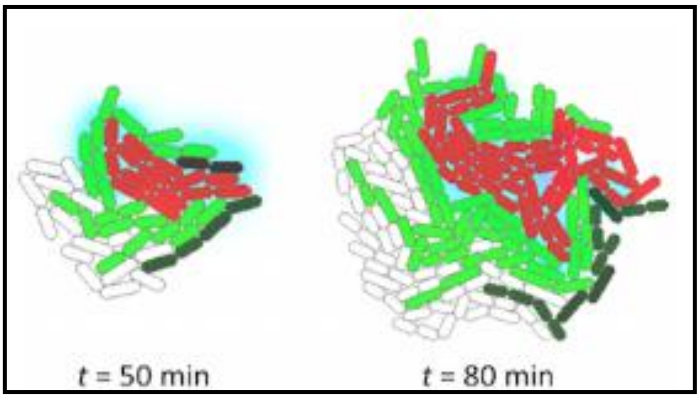


Figura 2.1 Ambiente de simulación en GRO mostrando un crecimiento poblacional bacteriano.

Figura disponible en: http://www.lia.upm.es/

La propuesta para mejorar las herramientas GRO enfocadas en reacciones entren metabolitos, corresponde a integrar modelos metabólicos pre entrenados por medio de Redes Neuronales Artificiales.

El aporte de este trabajo de investigación consistirá en la integración del vector de pesos (producto de modelo pre entrenado de Redes Neuronales Artificiales) dentro de las herramientas de GRO con la finalidad de que este vector calcule de forma sencilla el metabolismo celular de una bacteria para su simulación en ambientes poblacionales; sabiendo de antemano que una simulación de metabolismo en un instante de tiempo requerirá únicamente de computar una multiplicación de matrices (vector de metabolitos de entrada por vector de pesos) y no una resolución de ecuaciones diferenciales por medio de ecuaciones estequiométricas de la célula.

3 INTELIGENCIA ARTIFICIAL APLICADA A LA PREDICCIÓN DE FLUJOS METABÓLICOS CELULARES

3.1 Descripción del problema

La ingeniería metabólica es una disciplina que estudia las propriedades y características de los procesos metabólicos celulares, además de aplicar técnicas computacionales para el mejoramiento y control del crecimiento celular. El modelado de sistemas y simuladores poblacionales, representan una de estas técnicas aplicadas en la ingeniería metabólica, siendo el principal enfoque de aplicación de este trabajo.

El simulador poblacional de microorganismos GRO permite modelar, programar y especificar el crecimiento de microcolonias de microorganismos. Este simulador cuenta con funcionalidades y herramientas para representar redes metabólicas sencillas o complejas. Gracias a esta característica, la integración de ecuaciones estequiométricas que representan redes metabólicas se efectúa de forma programática, teniendo control del proceso computacional durante las simulaciones. Las funcionalidades de GRO y el acceso al código de programación nos brindan las herramientas necesarias para investigar en métodos eficientes para calcular flujos metabólicos de microorganismos.

El Balance de Flujo Metabólico (Flux Balance Analysis, FBA) celular es una técnica utilizada para determinar los flujos de reacción en una red metabólica. Este concepto nos brinda los datos necesarios para aproximar flujos metabólicos a expresiones matemáticas las cuales pueden ser descritas por código de programación, aportando a nuestro trabajo la facilidad y certeza de los datos experimentales. Además, aporta la versatilidad matemática de declaración por código en lenguajes de programación y por unidades de procesamiento de gráficos para acelerar el cálculo computacional.

Gracias a la efectividad y manejo de herramientas computacionales enfocadas en Inteligencia Artificial, podemos implementar estructuras de Redes Neuronales capaces de “aprender” modelos matemáticos. De esta forma, la Inteligencia Artificial jugará un papel importante en nuestra investigación, analizando formas computacionales para reducir tiempos de ejecución en cálculos de FBA en simuladores de microorganismos. Con base en los resultados, se presentará la propuesta de integración de las herramientas de Inteligencia Artificial en el núcleo de GRO.

La base del trabajo de investigación se centrará en poder resolver las problemáticas enunciadas en las siguientes secciones presentadas a continuación:

3.1.1 Evaluar si las técnicas de Inteligencia Artificial ofrecen mejores características que el modelo basado en ecuaciones estequiométricas.

Las características y cualidades de los algoritmos de Inteligencia Artificial proveen soluciones y mejoras a tareas específicas. No obstante, la complejidad de implementación, los resultados en predicciones, bases de datos para entrenamiento e incluso los resultados de predicción, pueden originar soluciones consideradas como no satisfactorias.

A lo largo de la realización de este proyecto, será de gran importancia analizar y encontrar soluciones ante problemáticas tales como:

- Entrenar un modelo de Red Neuronal con múltiples entradas y salidas, podría llegar a tardar días o semanas, con la posibilidad de no obtener resultados satisfactorios,
- Ejecutar una predicción de un modelo de Red Neuronal pre-entrenado, podría ser igual o mayor a la ejecución con respecto a FBA en COBRApy,
- Es posible que un modelo de Algoritmo Genético no satisface todas las soluciones,
- La selección, cruce y mutación en Algoritmos Genéticos es llevado a cabo con valores aleatorios, originando valores alejados de la realidad cuando se evalúan eventos no vistos antes,
- La mala determinación de las funciones de aptitud y evaluación pueden originar que el Algoritmo Genético nunca alcance la solución.

Por lo tanto, se deberá hacer un análisis comparativo entre los resultados obtenidos al aplicar algoritmos de Inteligencia Artificial y los sistemas basados en ecuaciones estequiométricas que implementa COBRApy. Con ello, se determinará qué método resulta más viable para su futura integración al simulador GRO.

3.1.2 Encontrar la función de aptitud en Algoritmos Genéticos

Una particularidad de los Algoritmos Genéticos es que cada problema tiene su propia solución y codificación, originando problemas para generalizar métodos de solución aplicando estas técnicas. Además, la optimización de funciones es dependiente de los parámetros implicados respecto a la función de evaluación y función de aptitud.

Un problema que enfrentan los Algoritmos Genéticos es la correcta definición de la función de evaluación ya que es la encargada de evaluar los resultados. Para lograr obtener la mejor población que represente la solución al problema planteado, se hace preciso estimar una fiabilidad numérica que evalué cada generación durante el proceso de selección. De esta forma es posible controlar cuantitativamente los mejores individuos durante la reproducción, cruce y mutación. Dentro del tema de investigación se hace complicada la definición de esta función debido al alto número de parámetros implicados durante el metabolismo celular del modelo *Escherichia coli* K-12 MG1655. En estas situaciones se puede hacer posible un estudio del comportamiento de las reacciones a diferentes concentraciones de metabolismos en la producción de biomasa.

La función de aptitud permite valorar la efectividad de la respuesta esperada, así como proveer un rango de error entre el valor esperado y el obtenido. La definición de esta función en nuestro campo de estudio se presenta en determinar los reactivos limitantes de las reacciones durante el metabolismo de la bacteria. Sabemos que los metabolitos limitantes son esenciales para la obtención de productos como la biomasa por lo que su determinación, será importante para controlar y optimizar los productos.

El análisis para definir estas funciones será visto en el capítulo 5 donde se someterá a prueba los respectivos métodos en su solución, para tomar una decisión respecto a su posible implementación en el simulador GRO.

3.1.3 Definir la arquitectura de Red Neuronal a utilizar

Lograr soluciones mediante Redes Neuronales está estrechamente vinculado con una de las topologías que permiten alcanzar el objetivo. Si bien, cada problema precisa de una solución en particular, existen métodos que se aproximan a la solución de problemas de clasificación, aproximación y búsqueda. Algunos de estos métodos son las Redes Neuronales Convolucionales para clasificadores de imágenes, Redes Neuronales usadas en aproximación y estimación.

La estructura de la Red Neuronal tiene gran impacto en la precisión de la solución y su capacidad de converger. La asignación de nodos y capas en la red origina la densidad de conexiones y a su vez, su capacidad para lograr una predicción cercana a la realidad. La problemática en juego resulta en estimar la cantidad de nodos y capas que han de utilizarse para resolver el problema planteado. Con falta de conexiones entre neuronas puede suceder que el algoritmo nunca encuentre una solución o tener un porcentaje de exactitud muy bajo. Por otra parte, al tener un número muy denso de conexiones la RNA presentará sobreajuste.

El modelo *Escherichia coli* K-12 MG1655 en la herramienta COBRApy cuenta con 74 metabolitos que intervienen durante un análisis de Balance de Flujo metabólico (FBA). Al tomar estos parámetros como datos entrada en la red neuronal origina más conexiones entre neuronas y capas, a su vez, ocasiona que aumente la densidad de conexiones, los tiempos de entrenamiento y a su vez, el tiempo de ejecución de un modelo pre-entrenado. Ante este punto es posible encontrar que los tiempos de ejecución de la RNA sean mayores o cercanos al método de FBA por medio de COBRApy.

3.2 Preguntas a resolver

Con la finalidad de encontrar soluciones a las problemáticas mencionadas anteriormente, se han planteado preguntas que ayudarán a satisfacer la implementación de algoritmos de Inteligencia Artificial capaces de ejecutar en menor tiempo el modelo FBA que aplica COBRApy. A continuación, se plantean las siguientes preguntas que ayudaran a resolver las problemáticas implicadas:

1. ¿Cuáles arquitecturas de Redes Neuronales Artificiales podrán aprender el modelo de Red Metabólica Celular para estimar FBA?
2. ¿Qué parámetros son esenciales para definir las funciones de aptitud y evaluación para estimar FBA por medio de Algoritmos Genéticos?
3. ¿Cómo influyen los metabolitos y sus límites de concentración en la creación de productos durante el metabolismo del modelo *Escherichia coli* K-12 MG1655?
4. ¿Es posible entrenar un modelo de Redes Neuronales Artificiales para estimar FBA con un mínimo de reactivos sin afectar su predicción?

En el capítulo 4 se describirán los métodos propuestos para hallar soluciones a las problemáticas y cuestiones implicadas en este trabajo de investigación. Analizando y estudiando los resultados obtenidos, se concluirá sobre la implementación en la práctica y trabajo a futuro.

4 APRENDIZAJE DEL BALANCE DE FLUJO METABÓLICO POR REDES NEURONALES

En el presente capítulo se desarrolla la metodología propuesta para entrenar un modelo de Redes Neuronales para la predicción del flujo metabólico celular. Usando como referencia el modelo *E. coli* K-12 MG1655 en la obtención de datos y costos computacionales, se pretende que el modelo de Red Neuronal aporte eficiencia en la predicción y menor costo computacional. Sin embargo, dado que existen múltiples estructuras de Red Neuronal vistas en la Sección 1.5.4, se ha optado por experimentar las estructuras de retro propagación ya que son de fácil abstracción e implementación en ambientes de programación.

A lo largo del presente capítulo se describirá la metodología utilizada para entrenar el modelo de Red Neuronal con el objetivo de encontrar resultados que satisfagan la problemática abordada. A fin de fundamentar los resultados obtenidos, se presenta un análisis de resultados que permitan comparar el tiempo y la eficiencia de predicción de la Red Neuronal implementada. Con ello se podrá evaluar su integración al simulador GRO.

4.1 Ambiente de Desarrollo

La base del desarrollo de programación e implementación se llevó acabo en las plataformas Pycharm 2019.2.3 y Jupyter Notebook, que proveen un ambiente de desarrollo completo para la programación en Python, además del soporte y comunidad en la integración de librerías. Ampliamente se han utilizado Keras, Pandas y la plataforma de TensorFlow que ofrecen herramientas de desarrollo en Redes Neuronales. A continuación, se muestran las versiones de las librerías y plataformas más utilizadas en el desarrollo de este trabajo:

- Jupyter Notebook: 4.61 /* Entorno de trabajo para desarrollo de código en Python */
- Pycharm: 2019.2.3 /* Entorno de desarrollo integrado para desarrollo de código en Python */
- TensorFlow: 2.0.0 /* Plataforma para desarrollo de modelo de aprendizaje automático */
- Cobra: 0.17.1 /* Librería enfocada al análisis metabólico basado en restricciones */
- Keras: 2.3.1 /* Estructura de código abierto para desarrollo de aprendizaje automático */
- scikit-learn: 0.22 /* Librería destinada al desarrollo de aprendizaje automático */
- Numpy: 1.17.4 /* Librería en Python destinada al análisis y visualización de datos */

A continuación, se mencionan las características del equipo de cómputo utilizado, además de ser el factor determinante de los tiempos de ejecución durante el desarrollo experimental de este trabajo.

- Sistema operativo: Debian 9,
- Procesador: Intel Core i5 – 4200 CPU 1.6 GHz,
- RAM: 4 Gbytes,
- Tarjeta de gráficos: Intel HD Graphics 4400,

4.2 Modelo de Referencia

Para llevar a cabo el desarrollo experimental de este trabajo se tomó en consideración el modelo metabólico *E. coli* K-12 MG1655, el cual ha sido expuesto en la Sección 2.2 de este trabajo. Este modelo de referencia de *E. coli* cuenta con 72 metabolitos y 95 reacciones y como se muestra en la Figura 4.1, el modelo se encuentra dentro del núcleo de trabajo de la librería COBRApy.

Con referencia en el Anexo B, en la Figura B se encuentra el núcleo de *E. coli* K-12 MG1655 y en la cual podemos encontrar la composición de los metabolitos y reacciones implicadas en su red metabólica. Es importante señalar que este modelo cuenta con 20 metabolitos extracelulares [15]. denotados por la etiqueta 'EX'.

Los metabolitos extracelulares conforman parte de los nutrientes que la célula puede absorber y producir del medio de crecimiento, por ello son utilizados en diversos estudios para control y análisis de crecimientos celulares, razón por la cual se utilizarán los metabolitos externos para estudio práctico de este trabajo.

```
In [1]: import cobra.test
        model = cobra.test.create_test_model("textbook")

In [3]: model

Out[3]:
```

Name	e_coli_core
Memory address	0x07fec49cc57b8
Number of metabolites	72
Number of reactions	95
Number of groups	0
Objective expression	1.0*Biomass_Ecoli_core - 1.0*Biomass_Ecoli_core_reverse_2cdba
Compartments	cytosol, extracellular

Figura 4.1 Modelo metabólico *E. coli* K-12 MG1655 dentro del núcleo de la librería COBRApy.

Como se ha mencionado, COBRApy cuenta con modelos metabólicos importados, destacando las estructuras metabólicas de Salmonella y *E. coli*. Este último se encuentra precargado por defecto y para trabajar con él se hace importando la sentencia:

cobra.test.create_test_model("textbook"), visible en la salida In [1] de la Figura 4.1.

Una vez cargado el modelo se puede hacer ejecución de la sentencia *optimize()*, la cual partiendo de las cantidades de nutrientes del medio de crecimiento es capaz de calcular el flujo metabólico celular. Se debe mencionar que para los modelos precargados como *E. coli*, la herramienta tiene asignados límites de concentración por defecto, pero de igual manera pueden ser reasignados por el usuario. En la Figura 4.2 se muestra la salida de la sentencia *model.optimize()*.

Así mismo, la sentencia *model.optimize()* despliega los valores de concentración de los reactivos implicados en el cálculo del flujo metabólico. Un aspecto sumamente importante es que dicha sentencia despliega el valor de la función objetivo que para el caso del modelo *E. coli* se refiere a la producción de biomasa. En particular se observa en la Figura 4.2 la leyenda **"Optimal solution with objective value 0.874"** que corresponde al valor de producción de biomasa igual a 0.874.

```
In [51]: #---Funcíon de resolucíon FBA---#
         model.optimize()
```

Out[51]: ***Optimal* solution with objective value 0.874**

	fluxes	reduced_costs
ACALD	0.000000	6.938894e-18
ACALDt	0.000000	0.000000e+00
ACKr	0.000000	1.040834e-17
ACONTa	6.007250	0.000000e+00
ACONTb	6.007250	1.387779e-17
...	...	...
TALA	1.496984	-1.387779e-17
THD2	0.000000	-2.546243e-03
TKT1	1.496984	-1.387779e-17
TKT2	1.181498	1.387779e-17
TPI	7.477382	-6.938894e-18

95 rows × 2 columns

Figura 4.2 Salida de la sentencia *model.optimize()*.

Entonces, cada vez que los valores de concentración de los reactivos hayan cambiado y se desee conocer la producción de biomasa y reactivos, se debe hacer uso de la sentencia *model.optimize()*.

4.3 Análisis muestral de FBA por modelo estequiométrico en COBRApy

En primera instancia, es necesario conocer el costo computacional que requiere el cálculo del flujo metabólico utilizando la librería COBRApy, considerando que este valor representa parte de los objetivos a cumplir en este trabajo y teniendo en cuenta que se busca que la Red Neuronal sea capaz de aprender a predecir el flujo metabólico y además minimizar el costo de procesamiento.

La primera muestra se realiza con los valores de los nutrientes por defecto del modelo *E. coli*. Con el propósito de obtener el tiempo de procesamiento promedio que requiere el cálculo de flujo metabólico en COBRApy, se utilizaron 200 iteraciones para obtener el valor medio y máximo de esta sentencia. En la Figura 4.3 podemos observar el código implementado, así como los valores siguientes:

```
In [55]: #---Tiempo de ejecución de FBA en Cobrapy---#
         optimizeList = [ ] #---Lista de iteraciones: optimize---#
         for i in range(200):
             star_time = time.time()
             model.optimize()
             end_time = time.time()
             timeOptimize = end_time - star_time
             optimizeList.append(timeOptimize)
         tiempoMedio =  round(np.mean(optimizeList),6)
         tiempoMaximo = round(np.max(optimizeList),6)
         print("Tiempo máximo de ejecucíon: ", tiempoMaximo)
         print("Tiempo medio de ejecucíon: ", tiempoMedio )

         Tiempo máximo de ejecucíon:  0.008186
         Tiempo medio de ejecucíon:  0.001794
```

Figura 4.3 Código para obtener el valor medio y máximo de la sentencia *model.optimize()*.

De forma gráfica se representan los 200 puntos de dispersión en la Figura 4.4, mostrando los tiempos de ejecución de la función *optimize()* en COBRApy. Se puede observar que la mayoría de los puntos se encuentran por debajo de los 0.002 segundos, localizando el punto medio en 0.001794 segundos y el punto máximo 0.008186 segundos. Gracias al valor medio de los tiempos de ejecución en COBRApy de la Figura 4.4 se puede inferir que ***calcular el flujo metabólico del modelo E. coli en COBRApy requiere de 0.001794 segundos.***

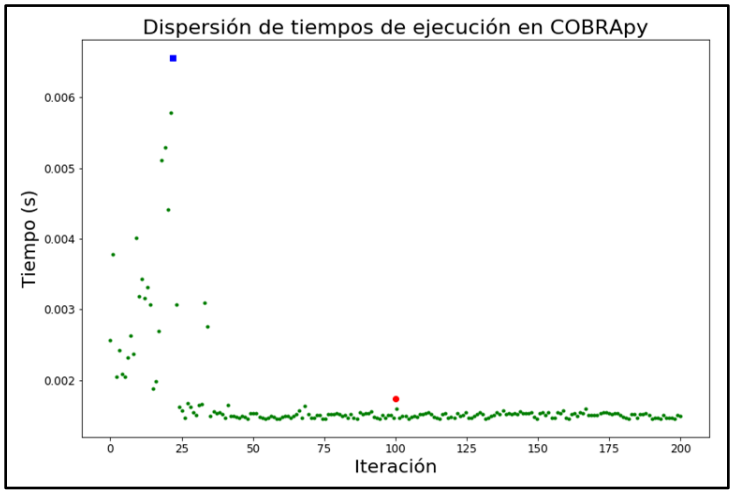


Figura 4.4 Dispersión de 200 tiempos de ejecución de la sentencia optimize().

4.4 Parámetros experimentales de entrenamiento para modelos de Red Neuronal

Analizar arquitecturas de Redes Neuronales que mejoren los tiempos de ejecución respecto a COBRApy es el principal enfoque en esta sección, además de buscar las respuestas a las preguntas planteadas en la Sección 3.5.

En primera instancia se requiere conocer el tiempo de ejecución de un modelo arbitrario de Red Neuronal que presente las mismas características del modelo *E. coli* en COBRApy.

Datos de entrada y salida

Los parámetros de entrada y salida de la arquitectura de Red Neuronal conforman los argumentos con los que opera COBRApy para calcular el flujo metabólico. Es decir, el modelo de Red Neuronal debe contar con lo siguiente:

- 20 datos de entrada, referente a los reactivos externos y
- 21 datos de salida, referente a los reactivos externos, más la función objetivo biomasa.

Los valores numéricos están determinados por las cantidades de concentración de cada reactivo y están sujetas al cumplimiento de las restricciones de estequiometria del modelo y la función objetivo. Las restricciones se denotan por los límites medidos en mmol/gDw/H (gramos de peso

seco de célula). COBRApy integra por defecto las concentraciones de los reactivos a 1000 y -1000 mmol/gDw/H, como se observa en la Figura 4.5 para la descripción del reactivo CO_2.

```
In [13]: model.reactions.EX_co2_e     #Descripción de la Reacción
```

Out[13]:

Reaction identifier	EX_co2_e
Name	CO2 exchange
Memory address	0x07fae8f1e0198
Stoichiometry	co2_e <=> CO2 <=>
GPR	
Lower bound	-1000.0
Upper bound	1000.0

Figura 4.5 Descripción del reactivo CO_2 en COBRApy.

4.4.1 Datos de entrenamiento

Los datos de entrenamiento deben brindar a la red la capacidad de converger y predecir datos de forma eficiente. En este caso de estudio se acordó que el conjunto de datos estará comprendido por 200 muestras que representen los 20 metabolitos [EX] en la capa de entrada. Si bien, no existe un valor máximo de concentración de nutrientes en un ambiente de crecimiento bacteriano, tomamos como medida de límite de concentración convencional de 0 a 30 mmol/gDw/H.

Para garantizar que las estructuras de Redes Neuronales están evaluadas bajo las mismas características, se ha de utilizar un único conjunto de datos de entrenamiento para evaluar las diferentes Redes Neuronales analizadas en este trabajo. Este conjunto de datos es creado en Python por medio de la librería *numpy* para 200 muestras aleatorias referenciando los 20 reactivos del modelo. El conjunto de entrenamiento es guardado para los futuros experimentos en formato CSV con el nombre 'DBEntrenamiento200.csv'

En la Figura 4.6 se muestra el conjunto de datos de entrenamiento utilizado, observando los diferentes reactivos con su límite inferior de cada evento.

```
In [7]: data.data
```

Out[7]:

	EX_acald_e	EX_ac_e	EX_akg_e	EX_co2_e	EX_etoh_e	EX_for_e	EX_fru_e	EX_fum_e	EX_glc__D_e	EX_gln__L_e	EX_glu__L_e
0	12.510660	21.609735	0.003431	9.069977	4.402677	2.770158	5.587806	10.366822	11.903024	16.164502	12.575835
1	24.022337	29.047847	9.402725	20.769678	26.291675	26.838200	2.551326	1.171643	5.094913	26.344275	2.950405
2	29.665833	22.444970	8.413320	23.678380	3.096780	13.436806	27.257865	8.808424	8.633260	3.900857	0.581009
3	3.070033	12.421680	20.832005	12.425378	1.498604	16.076892	19.913839	15.446673	28.337843	17.596651	27.102057
4	26.499183	18.710166	22.528273	10.466950	8.097837	26.876587	12.842736	28.945201	19.903245	18.650872	3.442379
...	...	...	...	...	...	...	...	...	...	...	...
195	29.252258	16.942038	13.972172	25.065760	6.598828	22.991943	5.760348	17.748827	18.204944	13.255314	25.936520
196	18.976058	5.611823	22.507319	16.465782	3.065111	0.744811	8.962386	20.700451	27.983651	28.994916	22.936443
197	13.466761	25.134858	5.392580	1.156772	21.980807	16.891222	1.261819	29.546558	17.079972	17.554700	23.697199
198	29.266259	22.911601	0.022227	22.056467	0.659775	5.056970	0.315420	4.473810	20.625166	21.060506	11.132322
199	24.004329	27.339251	15.394470	12.882661	2.747844	8.210319	26.599434	7.378330	18.635682	14.850222	2.831176

200 rows × 20 columns

Figura 4.6 Conjunto de datos de entrenamiento.

4.4.2 Función de activación

Como se ha mencionado, la función de activación se encarga de que la Red Neuronal tenga la capacidad de aprender de forma eficiente. La forma más común para medir el aprendizaje de la red es por medio del error absoluto medio (MAE) que expresa la diferencia entre el valor real de la medida y el valor que se ha obtenido a la salida de la red.

Durante el desarrollo experimental se utilizará MAE para medir la eficiencia de la predicción en las diferentes Redes Neuronales estudiadas. Sencillamente se busca que el error sea lo más cercano a cero para garantizar mejores predicciones a la salida de la red.

Si bien, la definición de las funciones de activación depende del comportamiento de los datos, en nuestro estudio contamos con datos dependientes y sujetos a una función objetivo por lo que es posible entrenar modelos con funciones de activación: RELU, ELU y SELU (descritas en la Sección 1.5.2). TensorFlow y Keras cuenta con soporte de integración de estas funciones, siendo útiles para el proceso de entrenamiento de modelos de aprendizaje por entrenamiento.

Uno de los primeros parámetros a considerar es encontrar la función de activación que mejor se adapta al problema planteado. Para lograrlo, se analizarán las curvas de aprendizaje para las funciones de activación: RELU, SELU y ELU. Los modelos estarán entrenados mediante el conjunto de datos generado en la Sección 4.5 y sujetos a un total de 100 épocas.

4.5 Análisis de Entrenamiento por función de activación

Con el fin de buscar el modelo de red neuronal que satisface los objetivos de la investigación, se requiere establecer una arquitectura de red que en un principio, sea capaz de aprender el modelo de predicción de flujo metabólico en COBRApy y que, a su vez, requiera de un menor tiempo de procesamiento en su cálculo.

En la elección del modelo se deben tomar en cuenta los datos de entrada y salida y un número de capas que pueda aprender de forma sencilla. Con un modelo de dos capas ocultas, se ha estudiado que puede ser capaz de aprender a razón del transcurso de las épocas de entrenamiento. Por lo tanto, el primer modelo estudiado comprende *una arquitectura de 20 neuronas de entrada, 2 capas ocultas con 10 neuronas por capa y 21 neuronas de salida*. Esta arquitectura se eligió en el equipo de trabajo y fundamentado por poseer una estructura de fácil implementación y que aportará resultados base para la investigación práctica. Además, esta estructura se encuentra entre los modelos de aprendizaje por retro propagación y aprendizaje profundo.

El primer modelo experimental describe la estructura de 20 neuronas de entrada 2 capas ocultas y 21 neuronas de salida. A partir de ahora será referenciado como modelo ‘MI20101021’ por poseer la siguiente estructura:

Modelo Inicial [MI] arquitectura: 20 – 10 – 10 – 21

Los experimentos para encontrar la función de activación que mejor se adaptan a la solución del problema permiten observar la curva de aprendizaje en las diferentes funciones y en qué medida se prevé que la red tendrá la capacidad de aprender. Además, nos proporcionará el tiempo de

ejecución promedio para 200 predicciones; buscar que sea menor 0.001794 segundos. Cabe mencionar que ejecutar una predicción conlleva a realizar una propagación hacia adelante de los datos de entrada y evaluar la salida.

4.5.1 Entrenamiento por función de activación Softmax

La Figura 4.7 muestra el comportamiento de la curva de entrenamiento para el modelo 'MI20101021' con Softmax (mencionada en la Sección 1.5.2) como función de activación. Se observa que el error después de 100 épocas es de 193.9387 por tanto, representa una predicción no eficiente en ningún sentido y después de las 100 épocas se estima que la red nunca converja.

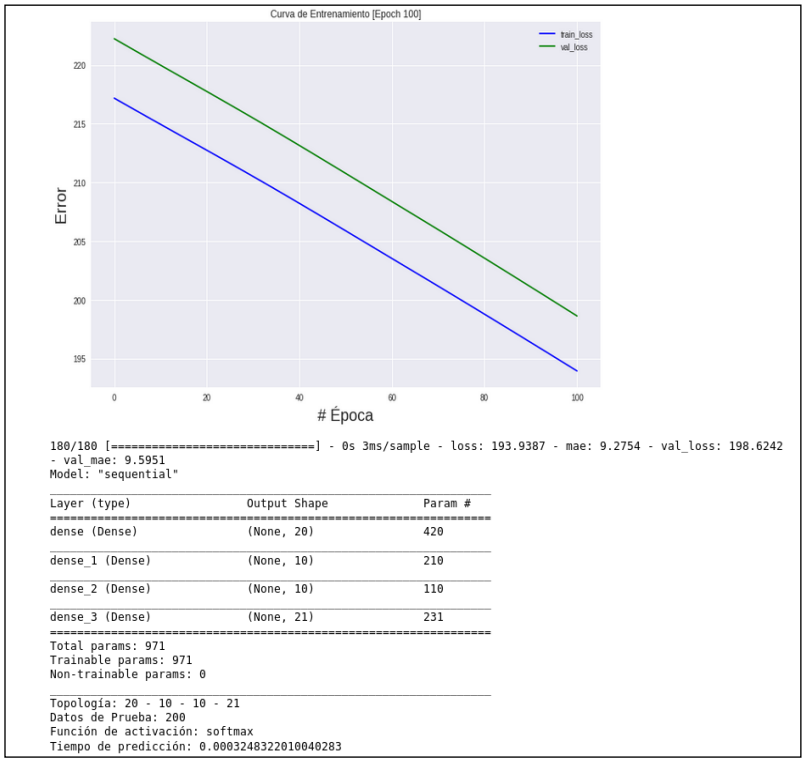


Figura 4.7 Curva de entrenamiento del modelo 'MI20101021' con función de activación Softmax.

Con este experimento se puede comprender el claro objetivo de activación Softmax al ponderar un valor a cada una de las salidas entre valores del 0 a 1, aquí se observa de manera clara la implementación de la función para problemas de clasificación. Para los experimentos futuros la activación Softmax queda descartada para su implementación.

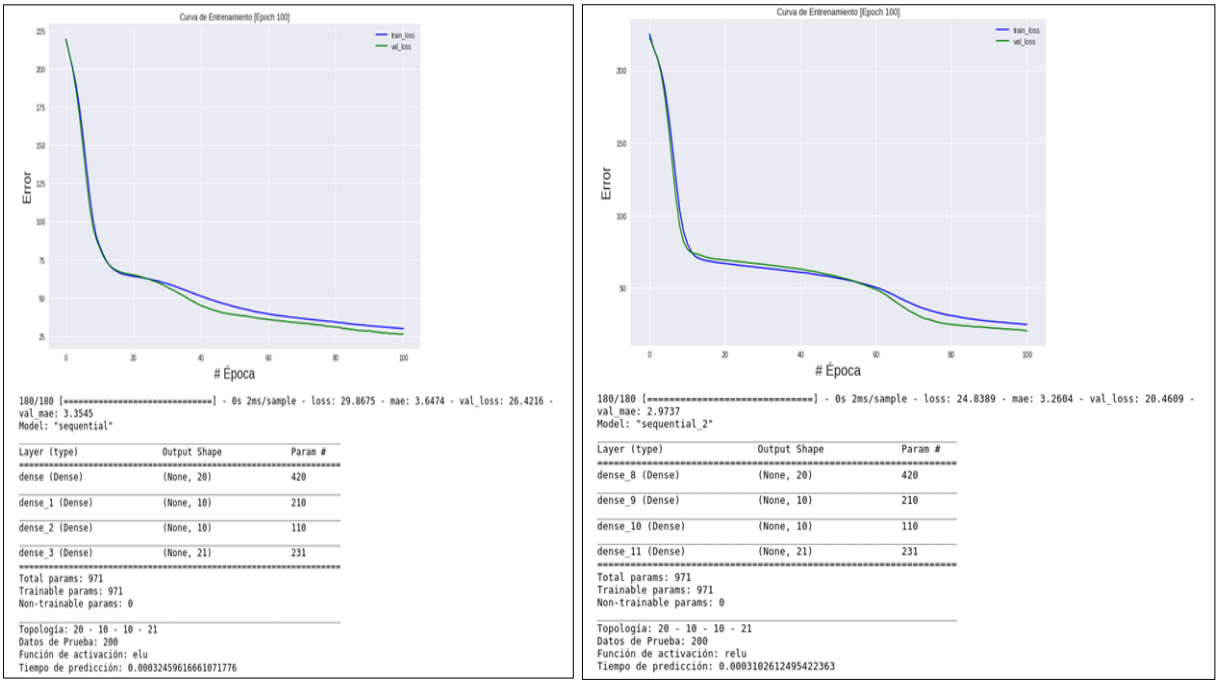
Al ser esta la primera red entrenada, arroja un valor importante respecto al tiempo de predicción. Se puede observar en la Figura 4.7 que el tiempo de predicción promedio tras doscientas muestras es igual a 0.0003248322 segundos, lo que demuestra que la arquitectura del modelo 'MI20101021' puede obtener tiempos de por lo menos 7 veces menor respecto a la ejecución de *model.optimize()* en COBRApy.

4.5.2 Entrenamiento por función de activación ELU y RELU

Las funciones de activación RELU, ELU y SELU (descritas en la Sección 1.5.2) son utilizadas para entrenar modelos con datos lineales y no problemas de clasificación como lo hace Softmax. A partir de los siguientes experimentos se entrenarán los modelos con funciones de activación para datos lineales.

En las Figuras 4.8.A y 4.8.B se observa el comportamiento de las curvas de entrenamiento del modelo 'MI20101021' ante las funciones de activación ELU y RELU. En primera instancia observamos que la curva de entrenamiento de la función ELU de la Figura 4.8.A presenta un error (loss) de 29.8675 después de 100 épocas. Sobre la primera época el error se encuentra sobre 220 mientras que pasadas 20 épocas el error se localiza sobre 70. Lo anterior indica que en las primeras 20 épocas el error se redujo en al menos 150 unidades, esto es un claro ejemplo de la integración de las funciones de activación aplicadas a datos lineales.

Un aspecto importante es el resultado de la predicción promedio igual a 0.0003245961 segundos, lo cual demuestra que la predicción de flujo metabólico por Redes Neuronales conlleva a menos costo de procesamiento respecto al obtenido con COBRApy en la Sección 4.6.



[A] [B]

Figura 4.8 Curva de entrenamiento del modelo 'MI20101021' con función de activación ELU [A] y RELU [B].

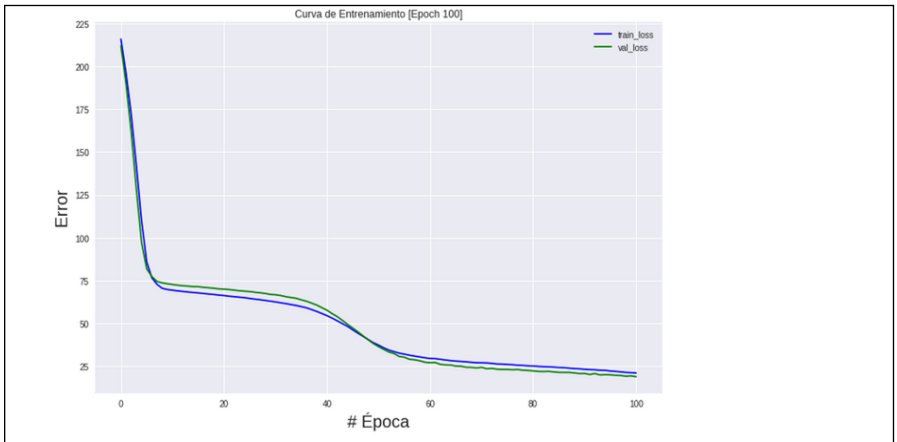
Respecto a la Figura 4.8.B observamos la curva entrenamiento de la función RELU, presentando similitudes respecto a la forma de la curva de la función ELU. Además, se obtiene un error de 24.8389, lo que resulta mejor en al menos 6 unidades respecto a la función ELU. Se puede agregar que entre la época 60 y 100 el error tiene un descenso de al menos 30 unidades, que representa que la red ajusta mejor los pesos y bias trascurridas a las 100 épocas respecto a la función ELU.

4.5.3 Entrenamiento por función de activación SELU

Los resultados obtenidos después de entrenar al modelo 'MI20101021' con la función SELU se muestra en la Figura 4.9. Se puede observar que al cabo de 100 épocas de entrenamiento el modelo ha obtenido un error (loss) de 21.0975. Este es resultado más cercano a cero de entre los 2 experimentos anteriores. Podemos afirmar que la función de activación SELU es la función que mejor se ha adaptado a los datos del problema y constituye la función de activación que se utilizará en los siguientes experimentos.

Respecto al tiempo de predicción [4], existe poca diferencia en comparación con las anteriores funciones. En la curva de entrenamiento de la Figura 4.9 se observa que para la época 80, donde el error se encuentra a 25 unidades, siendo un menor valor respecto a la función ELU. Llegando a este punto se puede observar que el error comienza en un ajuste a menor escala entre la época 80 y 100, teniendo un decremento de 6.0789 unidades.

El modelo de la Figura 4.9 obtiene 0.0003185009 segundos de predicción, aunque este valor no es el menor respecto a las 3 funciones de activación analizadas anteriormente, su valor resulta menor a los 0.001794 segundos obtenidos en COBRApy.




```
270/270 [==============================] - 0s 2ms/sample - loss: 21.0975 - mae: 2.9630 - val_loss: 18.9211 -
val_mae: 2.8527
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_40 (Dense)             (None, 20)                420
_________________________________________________________________
dense_41 (Dense)             (None, 10)                210
_________________________________________________________________
dense_42 (Dense)             (None, 10)                110
_________________________________________________________________
dense_43 (Dense)             (None, 21)                231
=================================================================
Total params: 971
Trainable params: 971
Non-trainable params: 0
_________________________________________________________________
Topología: 20 - 10 - 10 - 21
Datos de Prueba: 200
Función de activación: selu
Tiempo de predicción: 0.00031850099563598635
```

Figura 4.9 Curva de entrenamiento del modelo 'MI20101021' con función de activación SELU

De forma resumida, la Tabla 4.1 contiene los resultados obtenidos respecto al error y al tiempo de predicción obtenidos con las funciones de activación Softmax, RELU, ELU y SELU.

[4] Tiempo en realizar la propagación hacia adelante del vector de entrada de la red hacia el vector de salida y obtener un resultado.

Modelo 'MI20101021'

Topología	Función de activación	Error [MAE]	Tiempo medio de predicción a 200 muestras (segundos)
20 – 10 – 10 – 21	Softmax	193.9387	0.0003248322
	ELU	29.8675	0.0003245961
	RELU	24.8389	0.0003102612
	SELU	21.0975	0.0003185009

Tabla 4.1 Comparativa de tiempos de predicción y error de las funciones Softmax, RELU, ELU y SELU en el modelo 'MI20101021'.

Con el análisis anterior, se pueden establecer los siguientes puntos:

A) La función de activación SELU presenta el mejor ajuste de pesos y *bias* con los datos de entrenamiento, reduciendo el error en la predicción y, a su vez, obtienen un tiempo de predicción de al menos 7 veces menor respecto a COBRApy.

B) En primera instancia el modelo 'MI20101021' tiene un error por debajo de 20 unidades después del paso de 100 épocas; sin embargo, el error puede reducirse con más épocas de entrenamiento.

C) Modelos de Redes Neuronales con funciones de activación RELU, SELU y ELU, pueden aprender a predecir el flujo metabólico celular tomando como referencia datos de entrenamiento de la herramienta COBRApy.

4.6 Análisis del tiempo de predicción

Ahora que se conoce el tiempo de predicción del modelo de Red Neuronal, se probará la efectividad de la salida de la red con la idea de conocer si los resultados serán rápidos y con buena exactitud. Sabemos que la determinación de la exactitud dependerá de los datos de entrenamiento, las épocas y la arquitectura de red que mejor se adapta al problema.

Con el análisis de la arquitectura del modelo 'MI20101021' se obtuvieron resultados satisfactorios para considerar la integración de Redes Neuronales en la predicción del flujo metabólico celular; sin embargo, los experimentos anteriores se desarrollaron con la idea de conocer el tiempo de predicción sin enfocarse en la eficiencia en la salida de la red. En este punto resulta importante analizar qué parámetros afectan al tiempo y exactitud de los modelos de Red Neuronal implementados.

4.6.1 Primer análisis: Variando el número de neuronas en la primera capa oculta

Como se ha mencionado, el tiempo de entrenamiento no es un factor que precisa optimizar ya que una vez que la red aprende, realizar una predicción únicamente requiere de propagar los datos desde la capa de entrada hasta la capa de salida. Por lo tanto, el parámetro importante es el tiempo en que la red realiza una predicción.

Una idea planteada para saber cómo se ve afectado el tiempo de predicción respecto a la estructura de la red, es mediante la implementación de un algoritmo que obtenga los tiempos de predicción de diferentes estructuras de Red Neuronal. Tomando como referencia el modelo 'MI20101021', se propone desarrollar un algoritmo que defina, entrene y registre 100 épocas, el tiempo de predicción de modelos con 2 capas ocultas y un máximo de 50 neuronas en la primera capa oculta. De esta forma se podrán registrar los tiempos de predicción para cada una de las estructuras descritas y por medio de una gráfica de dispersión se podrán observar los resultados.

El algoritmo implementado se desarrolló respecto al software y al ambiente de trabajo descrito en la Sección 4.1. El pseudocódigo de la Figura 4.10 muestra la idea general del algoritmo llevado a cabo. De forma resumida, el algoritmo es capaz de definir, entrenar y evaluar Redes Neuronales de 0 a 50 neuronas en la primera capa oculta. Registrando el tiempo de propagación hacia adelante de los datos para su predicción.

```
Begin Tiempos de Predicción Respecto al Número de Neuronas por Capa
    - Crear conjunto de datos de entrenamiento: 'DBEntrenamiento200'
    - Crear conjunto de datos de evaluación: 'DBEvaluacion200'
    - Asignar máximo de neuronas por capa: maxNeuronas
    - Asignar estructura de red: arquitecturaRN [20, 0, 10, 21]
    - nNeuronas = 0
    - listTiemposDePrediccion = [ null ]
    - For maxNeuronas Do
        - Asignar estructura de red arquitecturaRN = [20, nNeuronas, 10, 21]
        - Declarar el modelo de red con Keras: model
        - Entrenar a 100 épocas el modelo con el conjunto de datos 'DBEntrenamiento200'
        - Evaluar model con los datos de evaluación 'DBEvaluacion200'
        - Insertar el tiempo de propagación hacia delante de model a la lista listTiemposDePredicción
    - Graficar por dispersión listTiemposDePredicción
    - Ubicar el tiempo mínimo y máximo mediante la gráfica de dispersión.
End
```

Figura 4.10 Pseudocódigo: tiempos de predicción respecto a neuronas por capa.

El resultado después de la ejecución del algoritmo se presenta en la Figura 4.11.

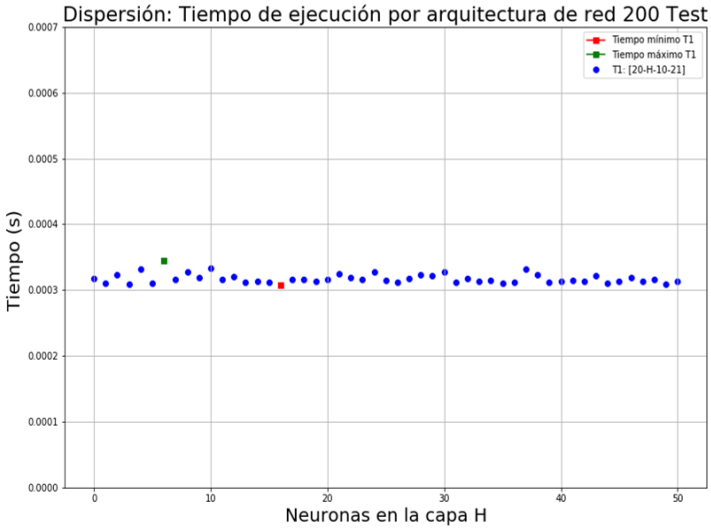


Figura 4.11 Dispersión de tiempos de predicción respecto al número de neuronas en la primera capa oculta, modelo [20 – H – 10 – 21].

En la Figura 4.11 se aprecia la secuencia de puntos que representan los tiempos de predicción de cada uno de los modelos creados por el algoritmo. El parámetro 'H' indica la cantidad de neuronas en la capa oculta correspondiente para cada evento. Como se puede observar, los tiempos están comprendidos entre 0.0003 y 0.0004 segundos, ubicado el mayor tiempo en 0.000345414 seg. y el tiempo mínimo de 0.000307945 segundos. Con estos datos podemos afirmar que la cantidad de neuronas en la primera capa oculta (sujeta a 10 neuronas en la segunda capa oculta), no afecta a gran escala el tiempo de predicción. Si bien, podemos encontrar similitud en tiempos cuando la red tiene 16 (tiempo mínimo de la Figura 4.11) y 49 neuronas en la capa H lo que indica que la cantidad de neuronas en la capa no influye en cierta medida.

El dato de importancia es el tiempo de predicción de la arquitectura 20 – 50 – 10 – 21 a 0.000313894 segundos. Lo anterior indica que a pesar de tener un mayor número de conexiones respecto al modelo 'MI20101021' el tiempo es de, al menos, 3 veces menor a una ejecución de COBRApy, permitiendo considerar entrenamientos con más de 100 épocas.

Hasta este punto consideramos evaluar modelos con diferentes cantidades de neuronas tanto en la primera y segunda capa oculta. Con ello se tendrán tiempos de predicción para estructuras que difieren en número de neuronas por capa.

4.6.2 Segundo análisis: Variando el número de neuronas en la primera y segunda capa oculta

En el segundo experimento el objetivo es conocer cómo varía el tiempo de predicción dependiendo de la cantidad de neuronas por capa para estructuras de Red Neuronal con dos capas ocultas, haciendo énfasis en conocer el tiempo máximo y minino de predicción en un rango de modelos dado.

La idea para satisfacer el objetivo del experimento es crear un algoritmo que entrene modelos para un mínimo y máximo número de neuronas por capas ocultas. Así, por ejemplo, se puede crear un

algoritmo que mediante ciclos aumente de forma ascendente el número de neuronas por capa hasta llegar a un punto máximo de neuronas donde en nuestro caso de estudio se han destinado 50 neuronas como límite. Así se puede registrar en cada modelo el tiempo de predicción; sin embargo, un algoritmo así entrenaría todos los modelos dentro del rango limitado. En otras palabras, si delimitamos un máximo de 50 neuronas por capa, el algoritmo entrenaría todo el rango de modelos comprendidos entre 20 – 0 – 0 – 21 hasta 20 – 50 – 50 – 21. Considerando lo anterior, el algoritmo tardaría un tiempo considerable en entrenar los modelos y es importante tomar en cuenta, como se ha visto en el experimento anterior, que el tiempo de predicción no varía a gran escala cuando variamos la cantidad de neuronas de una sola capa.

De acuerdo con lo mencionado y comprobado anteriormente, se dispuso a evaluar el tiempo de predicción para un intervalo de modelos menor. Primeramente, se registra el tiempo de predicción para los modelos de red cercanos a la estructura con menor cantidad de conexiones, para el rango comprendido entre 20 – 0 – 10 – 21 a 20 – 50 – 10 – 21 nombrado con la etiqueta T1. Posteriormente, se registran los tiempos de predicción de los modelos cercanos a la arquitectura con mayor número de conexiones, para el rango 20 – 50 – 0 – 21 a 20 – 50 – 50 – 21 ahora nombrado con la etiqueta T2.

Con los modelos T1 y T2 se cuenta con las arquitecturas máximas y mínimas respecto a la cantidad de conexiones, suponiendo por el momento, que los modelos 20 – 1 – 1 – 21 y 20 – 50 – 50 – 21, por la cantidad de conexiones, obtendrán los tiempos de predicción mínimos y máximos respectivamente. Tomando como referencia el pseudocódigo de la Figura 4.10 se dispuso a crear el algoritmo para evaluar los modelos T1 y T2 obteniendo los siguientes resultados.

Resultados

La gráfica de la Figura 4.12 muestra la dispersión de los tiempos de predicción para los modelos T1 y T2 observando que en general, los tiempos se encuentran dentro de los 0.0005 y 0.0003 segundos. Respecto al resultado de la gráfica, se localiza el tiempo mínimo en el modelo T1 igual a 0.000304404 segundos (punto rojo de la Figura 4.12) perteneciente a la estructura 20 – 13 – 10 – 21. Además, observamos que la estructura 20 – 50 – 50 – 21, que posee el mayor número de conexiones del rango, obtiene un tiempo de 0.000408709 segundos.

Es preciso mostrar que, el modelo 20 – 50 – 50 – 21 es el que cuenta con mayor número de conexiones y, sin embargo, este no ha obtenido el mayor tiempo de predicción, sino que el menor tiempo registrado pertenece al modelo 20 – 50 – 39 – 21. Con este resultado se observa que el tiempo de predicción no depende en gran escala de la cantidad de neuronas cuando aumentamos la cantidad en una única capa, sino que existe un ajuste por la composición de neuronas por ambas capas ocultas.

Estos resultados son importantes ya que demuestran que una arquitectura de red con 2 capas ocultas con un máximo de 50 neuronas por capa puede obtener tiempos de predicción menores al de FBA en COBRApy. Además, este hecho indica que podemos entrenar de forma eficiente modelos con más de 100 épocas y entregarán tiempos eficientes de predicción.

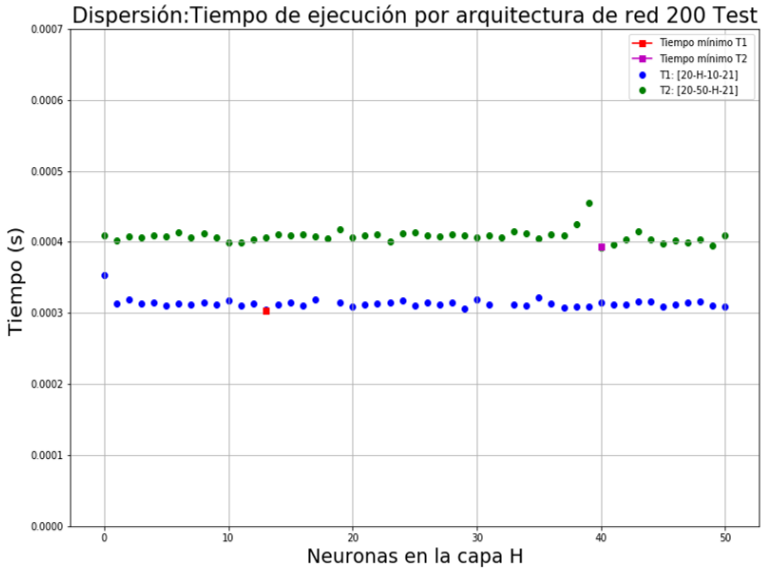


Figura 4.12 Dispersión de tiempos de predicción para los modelos T1[20-H-10-21] y T2[20-50-H-21].

Al poder medir los tiempos de la estructura T1 y T2 se puede notar que los tiempos de predicción de la estructura T2 se encuentran sobre los 0.0004 segundos. Entonces, la estructura T2 cuenta con 50 neuronas en la primera capa oculta y este valor no cambia a medida que aumenta la cantidad de neuronas de la segunda capa. Por tal razón, la estructura T2 contará con mayor número de conexiones en todo momento respecto a los modelos en T1. En este punto podemos afirmar que ***el tiempo de predicción es dependiente del número de conexiones, a mayor cantidad de conexiones, mayor es el tiempo de propagación hacia adelante.***

4.6.3 Tercer análisis: Variando el número de capas ocultas

Gracias a los dos experimentos realizados hasta ahora se ha visto cómo varía el tiempo de predicción con el cambio en la cantidad de neuronas en una sola capa, permitiendo observar que el tiempo no cambia a gran escala con el aumento del número de neuronas en una capa.

Ahora es preciso conocer el tiempo de predicción para modelos con mayor número de capas ocultas respecto al modelo 'MI20101021'. Dicho lo anterior, se optó por comparar modelos con 5 capas ocultas, registrando y analizando la comparación de los tiempos de predicción respecto a los modelos con 2 capas ocultas.

Con este análisis se podrá saber si podría resultar viable la implementación de estructuras con 5 capas ocultas, ya que como se ha observado en la Figura 4.12, a mayor número de neuronas en la red, el tiempo de predicción tiende a aumentar.

En este experimento entrenamos y analizamos los tiempos de predicción para 200 datos de evaluación con las topologías mencionadas a continuación. Se utiliza el procedimiento descrito en la segunda prueba (Sección 4.6.2). Los modelos con dos capas ocultas se representan con la etiqueta T1 y aquellos con cinco capas ocultas se etiquetan como T2. A continuación, se presentan los conjuntos de redes entrenadas para los modelos T1 y T2.

- T1: 20 – 0 – 10 – 21 a 20 – 50 – 10 – 21
- T2: 20 – 0 – 50 – 50 – 50 – 50 – 21 a 20 – 50 – 50 – 50 – 50 – 50 – 21

Modelos analizados durante la tercera prueba

Resultado

Realizado el entrenamiento de las estructuras T1 y T2, se obtuvo la gráfica de dispersión de la Figura 4.13. Con este experimento comparamos estructuras con 2 y 5 capas ocultas. Como resultado de la Figura 4.13 podemos observar una diferencia de al menos 0.0002 segundos entre ambas topologías. Lo sobresaliente de este experimento es el tiempo mínimo de la topología con 5 capas ocultas que obtiene un tiempo mínimo de 0.000509989 segundos en la topología 20 – 43 – 50 – 50 – 50 – 50 – 21, este dato representa un hecho importante ya que sabemos que, a pesar de que los modelos de 5 capas ocultas tienen mayor número de conexiones respecto al modelo 'MI20101021', que los tiempos de predicción siguen siendo menores respecto a COBRApy. Siendo lo anterior, uno de los principales objetivos que busca este trabajo.

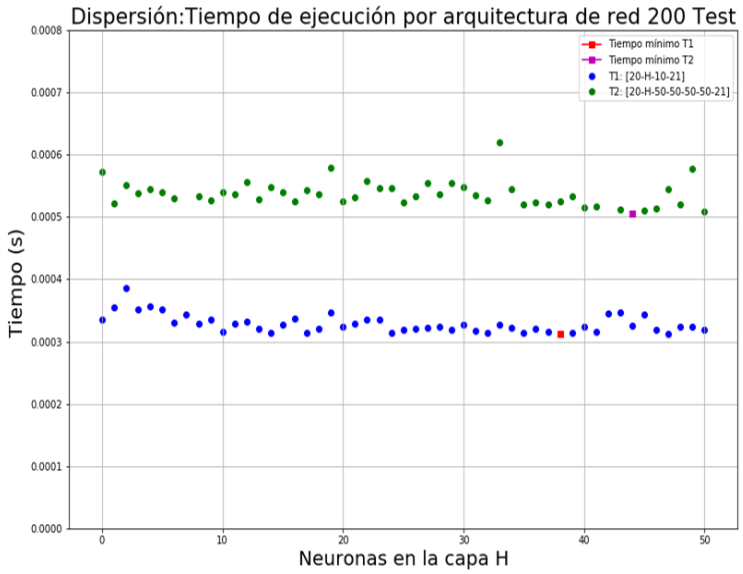


Figura 4.13 Dispersión de tiempos de predicción para los modelos T1[20-H-10-21] y T2[20-H-50-50-50-50-21].

Lo presentado en la Figura 4.13, permite conocer que los modelos de 2 y 5 capas ocultas obtienen tiempos de predicción entre los 0.0003 y 0.0006 segundos, siendo este dato de importancia ya que mejora la ejecución de flujo metabólico con respecto a COBRApy. Este análisis permite tomar en cuenta cualquiera de los modelos T1 y T2 para entrenar con mayor número de épocas respecto a las 100 épocas integradas en los experimentos anteriores.

Con estos tres experimentos hemos podido conocer que las arquitecturas de red con 5 capas ocultas y 50 neuronas por capa obtienen tiempos de predicción de FBA menores a COBRApy, lo que da los fundamentos para considerar la integración de Redes Neuronales predictoras de FBA en simuladores de colonias de bacterias.

Hasta ahora hemos estudiado el tiempo de predicción sin tomar en cuenta el error a la salida de la Red Neuronal.

Como se mencionaba anteriormente, los modelos estudiados fueron entrenados a 100 épocas cada uno siendo el principal objetivo conocer el tiempo de predicción antes que la eficiencia de los datos a la salida de la red. Añadiendo también que la eficiencia no se registró, ya que se supone que los modelos obtendrán un error alrededor de 21, tal como lo visto en la Sección 4.7 con el modelo 'MI20101021'.

4.7 Análisis del error mínimo

Con los experimentos analizados en la Sección 4.6 sabemos que el tiempo de predicción con Redes Neuronales es eficiente, resultando sumamente importante estudiar los parámetros que afectan el error a la salida de la red. En la Figura 4.8 se analizó la curva de aprendizaje del modelo 20 - 10 10 - 21 donde se puede determinar que el error se reduce a medida que el número de épocas se acerca a 100; este hecho hace evidente que, para un número mayor de épocas, el modelo hará predicciones con menor error.

En este estudio experimental se busca conocer el valor de un conjunto de topologías con dos capas ocultas cuando son entrenadas con 500 épocas. El experimento permitirá conocer cómo se ajusta el error a mayor número de capas. En este experimento el tiempo de predicción no se tomará en cuenta debido a lo observado en la Figura 4.11, donde los tiempos de predicción se encuentran entre 0.0003 y 0.0004 segundos, haciendo eficiente el tiempo de predicción en su implementación.

4.7.1 Análisis de error con 2 capas ocultas

Durante la experimentación se utilizará el conjunto de topologías de dos capas ocultas y un máximo de 80 neuronas por capa, por lo tanto, la topología de red con menor número de conexiones es 20 – 0 – 0 – 21 y la topología 20 – 80 – 80 – 21 con mayor número de conexiones. Cada modelo ahora es entrenado con 500 épocas y evaluando con 200 datos. Al término de este algoritmo obtendremos una gráfica de dispersión que visualizará el error respecto al número de neuronas en cada capa. Al final, obtendremos el modelo que brinda el menor error y su respectivo tiempo de predicción.

Resultado

Producto del entrenamiento de los modelos de arquitectura antes mencionados, se obtuvo la gráfica de la Figura 4.14, que muestra la dispersión del error respecto al número de neuronas por cada capa. Con base en esta gráfica se puede observar cómo el error disminuye a medida que el número de neuronas por capa aumenta. También se puede observar cómo el error disminuye significativamente cuando el número de neuronas por capa es mayor a 30.

Podemos observar una tendencia a disminuir gradualmente del error para los modelos con más de 10 neuronas por capa. Así, continúa disminuyendo el error gradualmente hasta cuando existen 17 neuronas por capa, donde el error es aproximado a 4 para la estructura 20 – 17 – 17 – 21. Podemos apreciar cómo el error ya no disminuye en gran medida dentro de las estructuras de más de 30

neuronas por capa y hasta 68 neuronas por capa (aproximadamente disminuye en 3 unidades). Para los modelos con más de 68 neuronas por capa, se observa que el error comienza a acercarse a cero.

Observando la Figura 4.14 podemos decir que los modelos de red dentro del conjunto de 70 a 80 neuronas por capa oculta obtendremos un error similar en cualquiera de ellos, lo que indica que estos modelos presentan una predicción eficiente a la salida de la red; no obstante, los modelos con menor número de neuronas, por ejemplo 20-45-45-21, también presentan valores de error cercano a cero y es probable que estos conjuntos, al cabo de más épocas, podrán ajustar sus errores cercanos a cero, sólo que en este experimento se busca cuáles modelos ajustan más rápido y más preciso el error a la salida de la red.

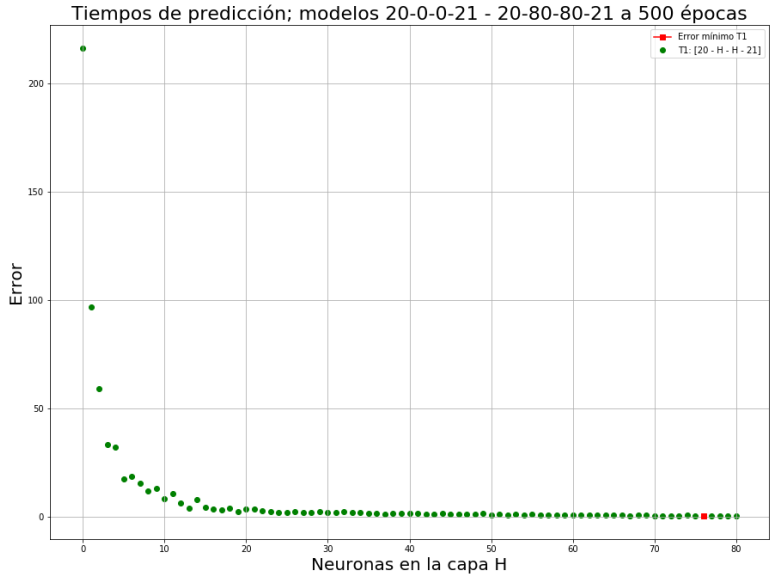


Figura 4.14 Dispersión de tiempos de predicción para el conjunto de modelos 20-0-0-21 - 20-80-80-21

Como dato importe, el modelo 20-76-76-21 (marcado con punto rojo en la Figura 4.14) representa el mínimo error dentro del conjunto estudiado, donde es igual a 0.344210, además de obtener un tiempo de predicción de 0.000321653 segundos. Este dato resulta ideal para implementarlo como modelo final para su puesta en marcha dentro del simulador GRO. En resumen, este modelo proporciona un tiempo de al menos 3 veces menor del modelo de FBA de COBRApy y los datos de predicción son semejantes a la herramienta COBRApy con un error de 0.344210.

Con fines prácticos el modelo representativo al error mínimo es etiquetado como ‘EM20767621’, haciendo referencia al Error Mínimo y a su arquitectura de capas: 20 – 76 – 76 – 21. La Tabla 4.2 presenta los parámetros del modelo ‘EM20767621’.

Parámetros del modelo “EM20767621”

Arquitectura	Datos de entrenamiento	Épocas	Función de activación	Error	Tiempo de predicción
20 – 76 – 76 – 21	200	500	SELU	0.344210	0.000321653

Tabla 4.2 Parámetros del modelo “EM20767621”.

Los resultados encontrados en este experimento han sido satisfactorios respecto a que el error disminuye al paso de las épocas; sin embargo, existe la posibilidad de que los modelos antes experimentados no lleguen a disminuir aún más el error a través del paso de las épocas. Este hecho podrá suceder debido a que la cantidad de capas ocultas no es lo suficiente para disminuir el error.

4.7.2 Análisis de error con 3 capas ocultas

En este experimento se realiza un algoritmo con el mismo principio de la sección anterior, pero únicamente agregando una capa oculta. Entrenamos Redes Neuronales con 3 capas ocultas y hasta 80 neuronas por capa, lo que evaluará el conjunto de arquitecturas: 20 – 0 – 0 – 0 – 21 a 20 – 80 – 80 – 80 – 21.

Resultado

El entrenamiento de los modelos de arquitecturas 20 – 0 – 0 – 0 – 21 a 20 – 80 – 80 – 80 – 21 son presentados en la Figura 4.15 con un tiempo de entrenamiento de 2 horas. Principalmente se puede destacar la forma en que el error se acerca a cero ocurriendo a partir del modelo con 27 neuronas por capa. Realizando una comparación con las Figuras 4.14 y 4.15, se observa qué las estructuras de 3 capas ocultas el ajuste del error se acerca a cero cuando los modelos tienen más de 70 neuronas por capa. Se puede decir entonces que los modelos con más de 3 capas ocultas tienden a ajustar los pesos más rápido respecto a aquellos con dos capas ocultas.

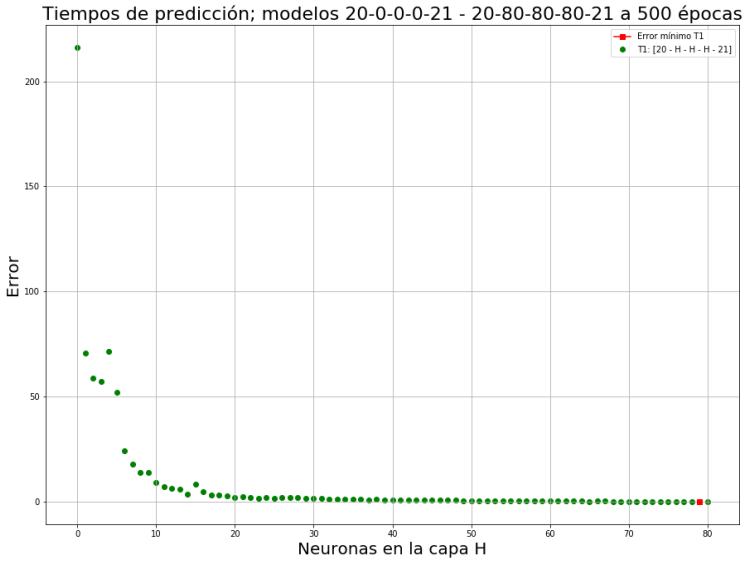


Figura 4.15 Dispersión de tiempos de predicción para el conjunto de modelos 20-0-0-0-21 - 20-80-80-80-21.

Nuevamente podemos ubicar el modelo que presenta el error mínimo dentro del conjunto estudiado, estando presente en la estructura 20-79-79-79-21 con un error mínimo de 0.091635 y un tiempo de predicción igual a 0.000488706 segundos. Este modelo es etiquetado para referencias futuras como “EM2079797921”, la Tabla 4.3 presenta los parámetros del modelo.

Parámetros del modelo "EM2079797921"

Arquitectura	Datos de entrenamiento	Épocas	Función de activación	Error	Tiempo de predicción
20 – 79 – 79 – 79 – 21	200	500	SELU	0.091635	0.000488706

Tabla 4.3 Datos del modelo "EM2079797921".

Comparando los modelos de las Tabla 4.2 y 4.3 podemos decir que el menor tiempo se presenta en el modelo de 2 capas ocultas y el menor error en el modelo de 3 capas ocultas.

En este punto, es importante considerar que si bien, el error y el tiempo de predicción no difieren en gran cantidad entre modelos de 2 y 3 capas ocultas, ambos pueden ser llevados a la implementación basándose en el objetivo de esta investigación, ya que ambos mejoran el tiempo de ejecución de FBA en COBRApy. Entonces, con esta idea podemos establecer que, si de forma práctica requerimos de disminuir el tiempo de predicción de FBA en simuladores, podemos implementar la red neuronal 'EM20767621' y si requerimos que los valores de salida sean lo más cercanos a los datos reales, integraremos modelos de 3 capas ocultas tal como el modelo 'EM2079797921'.

4.8 Mejoras en tiempo y error

Un aspecto interesante en los dos últimos experimentos es que los resultados fueron analizados a 500 épocas, indicando que pueden obtenerse mejores resultados de error a más épocas de entrenamiento. Entonces, se hace oportuno comparar la curva de entrenamiento de los modelos 'EM20767621' y 'EM2079797921' después de 500 épocas.

Para este último experimento comparamos la curva de entrenamiento de los modelos 20-76-76-21 y 20-79-79-79-21 con 4000 épocas. En este experimento obtendremos el error mínimo ente los dos modelos después de las 4000 épocas y podemos analizar la diferencia entre ambos, para determinar cuál de ellos puede llevarse a la implementación en el simulador GRO.

Resultados

Mediante la Figura 4.16 observamos la curva de entrenamiento de los modelos 'EM20767621' y 'EM2079797921'. Este experimento se culminó en 30 minutos.

La curva de entrenamiento de los modelos se observa en la Figura 4.16 presentando una gran similitud en las curvas de ambos modelos. El resultado de los datos analizados se encuentra en la Tabla 4.4.

Modelo	Datos de entrenamiento	Épocas	Función de activación	Error	Tiempo de predicción
EM20767621	200	4000	SELU	0.010466	0.000312676
EM2079797921	200			0.002741	0.000350507

Tabla 4.4 Datos del modelo 'EM20767621' y 'EM2079797921'.

Los datos de la Tabla 4.4 nos dicen que existe una diferencia de error entre los modelos de al menos 0.01 unidades, siendo un valor de tolerancia aceptado. Entonces podemos decir que ambos modelos tienen un error y tiempo de predicción muy aceptable para su integración en los simuladores y que cumple eficientemente con el objetivo principal de este trabajo de tesis.

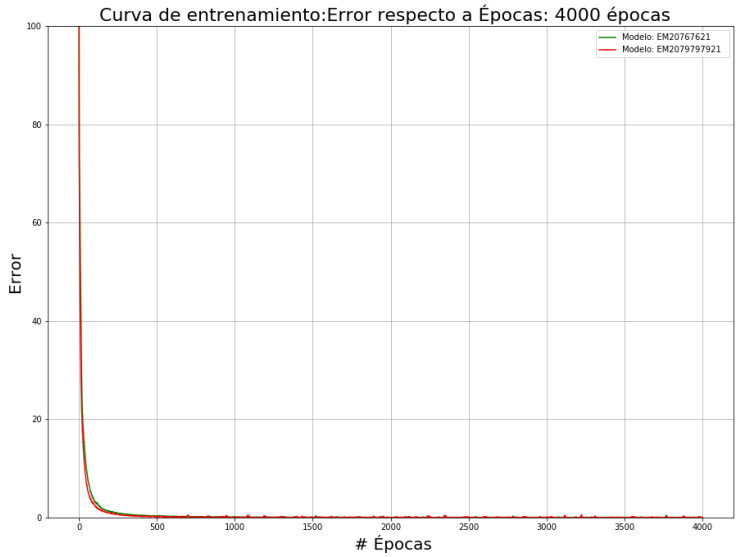


Figura 4.16 Comparación de curva de entrenamiento a 4000 épocas de los modelos ‘EM20767621’ y ‘EM2079797921’.

Tomando en consideración los resultados, podemos afirmar que el modelo ‘EM20767621’ puede ser entrenado con un número mayor de épocas, existiendo una probabilidad de que en algún punto el error de ‘EM20767621’ sea igual a ‘EM2079797921’ e incluso pueda llegar a ser mejor; sin embargo, este escenario es impredecible, a menos que se someta a entrenamiento el modelo ‘EM20767621’ hasta llegar a un error mayor al del modelo ‘EM2079797921’.

Para fines prácticos y valiéndose de los objetivos y principios de esta investigación, podemos concluir que:

“Con el modelo de red neuronal 20 – 76 – 76 – 21 se obtiene un tiempo de predicción de 0.000312676 segundos para realizar un cálculo de flujo metabólico del modelo E. coli K-12 MG1655, siendo mejor el modelo predictivo por Redes Neuronales que el modelo presentado en la herramienta COBRApy”

Es importante mencionar que tanto el modelo ‘EM2079797921’ como el ‘EM20767621’ no son los únicos que pueden implementarse en el simulador ya que, como se vio en las Figuras 4.14 y 4.15, existe un gran número de modelos de 2 y 3 capas ocultas que pueden llegar a resultados muy similares. Entonces, este modelo ha sido resultado de los casos de prueba, análisis y resultados obtenidos durante la investigación y son sujetos a los datos de entrenamiento creados en la Sección 4.4.1, lo que comprueba que con otro conjunto de datos de entrenamiento se pueden obtener otros resultados.

4.9 Validación de resultados

Como último paso se dispuso a evaluar y validar la Red Neuronal con un conjunto de datos de prueba. En este experimento se pasará un conjunto de 20 datos de entrada referentes a los 20 metabolitos externos en el medio, con un límite de concentración de 0 a 20. A modo de validación, se compara el mismo conjunto de datos entre COBRApy y el modelo de Red Neuronal 'EM20767621'.

La Figura 4.17 muestra un total de 100 datos de validación para los modelos 'EM20767621' y COBRApy, representando en columnas los metabolitos externos. La salida de la sentencia Out [85] en la Figura 4.17 muestra los datos resultantes al calcular el flujo metabólico con el modelo 'EM20767621' mientras que la Out[86] presenta los datos obtenidos en COBRApy.

De forma práctica se muestra la Tabla 4.5 que tiene la finalidad de comparar los resultados de la función objetivo biomasa. La tercera columna muestra el error de la salida de la Red Neuronal respecto a la salida en COBRApy haciendo alusión a la diferencia de la salida obtenida respecto a la salida esperada. Se puede apreciar que la diferencia en algunos casos es mínima. Tomando como ejemplo la salida del caso 4, se obtiene un error de 0.010966.

Con los resultados obtenidos se puede afirmar que el modelo de Red Neuronal ha "aprendido" a calcular el flujo metabólico y ofrece mayor velocidad de cálculo de reacciones metabólicas. También podemos considerar el modelo de Red Neuronal para estudios de procesos metabólicos para otros agentes microbianos o modelos de prueba. Una gran ventaja que ofrece nuestro modelo es la simplicidad de su integración como un conjunto de valores numéricos integrados en vectores y matrices. Estos valores numéricos corresponden a los pesos y *bias* que el modelo ha encontrado para satisfacer las predicciones de forma eficiente. Además, su implementación e integración en modelos computacionales biológicos o simuladores de microorganismo puede llevarse a cabo mediante la asignación de pesos y *bias* que se encuentran guardados por la librería Keras. Los anteriores datos pueden ser guardados en formato csv[5] para su integración en lenguajes de programación.

Es importante considerar que si bien, al igual que otras arquitecturas de Red Neuronal pueden obtener resultados muy similares, estos resultados siempre dependerán de las épocas, los datos de entrenamiento y su estructura, aportando diversos resultados para su estudio. En propósito de la investigación y el tiempo de estudio, los resultados anteriores han satisfecho las necesidades y problemáticas planteadas en este trabajo.

[5] csv (comma-separated values) son archivos comúnmente utilizados en bases de datos para representar información en formato de columnas y saltos de línea.

In [85]:
```
print("Datos de Validación FBA Red Neuronal 'EM20767621'")
dataOutputNN
```

```
Datos de Validación FBA Red Neuronal 'EM20767621'
```

Out[85]:

	EX_acald_e	EX_ac_e	EX_akg_e	EX_co2_e	EX_etoh_e	EX_for_e	EX_fru_e	EX_fum_e	EX_glc__D_e	EX_gln__L_e	...	EX_h2o_e	EX_h_e
0	0.659714	10.170858	28.681877	10.981362	-14.406718	-0.187989	-0.046629	-10.617404	-0.117722	-5.376800	...	21.285280	8.22633
1	8.109811	1.946894	3.209845	17.195147	-16.681334	-0.903341	-1.073316	-1.481169	0.158541	-7.766757	...	25.005293	2.49114
2	5.587628	9.124235	6.368937	16.624794	-27.040010	1.409795	-0.392109	-0.545515	-0.123459	-16.798197	...	29.427031	16.4320
3	-12.748221	5.950713	7.536399	3.734187	7.464687	-0.927633	-1.596437	-0.294152	-0.452431	-1.788300	...	2.638902	8.89102
4	-8.673592	2.453430	10.006711	15.316653	-0.235059	0.324359	-1.553907	-1.483925	0.045322	-2.279765	...	27.213917	7.55207
...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	-18.568535	12.649332	13.384475	6.208229	-8.439894	-0.504536	-1.844840	-5.472435	-0.100815	-0.017078	...	22.279640	26.9611
96	-1.207369	13.700267	3.279450	19.508856	-28.073942	-0.087012	0.245674	-0.715234	-0.397889	-12.681305	...	23.505892	20.0007
97	-23.108728	8.036126	2.495161	29.461811	7.733160	0.130662	-5.095778	0.852827	0.395961	0.146155	...	19.044029	16.6024
98	7.253517	5.491243	13.878462	21.855350	-18.613071	-0.339746	-0.785174	-5.050213	0.168505	-4.116801	...	27.438189	4.11675
99	-13.202416	9.845574	7.995610	5.849050	2.542990	1.633632	0.079718	-10.043995	-0.392276	-4.109011	...	4.160817	6.33269

100 rows × 21 columns

In [86]:
```
print("Datos de Validación FBA COBRApy")
test.value
```

```
Datos de Validación FBA COBRApy
```

Out[86]:

	EX_acald_e	EX_ac_e	EX_akg_e	EX_co2_e	EX_etoh_e	EX_for_e	EX_fru_e	EX_fum_e	EX_glc__D_e	EX_gln__L_e	...	EX_h2o_e	EX_h_e
0	6.42521	11.72895	27.05857	11.72565	-22.45394	0.0	-0.00000	-13.77550	0.0	-7.41650	...	20.63600	7.83074
1	0.91847	3.26865	1.05234	17.70153	-8.32123	0.0	0.00000	0.00000	0.0	-2.19962	...	25.05867	1.71477
2	3.49284	10.92789	6.76043	18.32727	-24.93307	0.0	-0.23773	0.00000	0.0	-15.75448	...	29.08169	15.68471
3	-15.79037	7.24527	9.52043	2.34368	7.26733	0.0	-0.00000	-2.95283	0.0	-0.08718	...	1.80225	7.22767
4	-11.90397	2.45002	10.29866	15.93076	5.01507	0.0	-1.10104	0.00000	0.0	-0.28304	...	26.35448	7.82757
...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	-12.16475	10.33778	9.30075	5.51977	-10.80270	0.0	-3.97681	-5.07980	0.0	0.00000	...	22.67593	28.01779
96	-2.13403	13.91312	3.72635	20.02443	-25.16278	0.0	0.00000	0.00000	0.0	-13.94573	...	24.44975	18.94630
97	-34.08623	6.84323	3.32762	29.06464	6.32437	0.0	-1.47871	0.00000	0.0	-0.38012	...	19.50825	16.59948
98	1.38129	4.12583	13.18167	21.13859	-9.86631	0.0	0.00000	0.00000	0.0	-0.29741	...	23.86294	2.53121
99	-19.84380	9.25310	10.33894	7.00866	8.52318	0.0	0.00000	-7.99431	0.0	-0.14106	...	5.71087	6.00822

100 rows × 21 columns

Figura 4.17 Datos de validación entre la librería COBRApy y el modelo de Red Neuronal ‘EM20767621’.

Si bien, hasta este punto no podemos decir que el modelo encontrado ‘EM20767621’ representa una solución global al problema, nos proporciona la forma para generar más ideas para desarrollar algoritmos que optimicen una arquitectura de Red Neuronal que se ajuste a nuestras necesidades. Se puede plantear un trabajo futuro para optimizar arquitecturas creadas mediante algoritmos genéticos.

Out[82]:

	Biomasa_FBA_COBRApy	Biomasa_FBA_ME20797921	Error
0	1.14728	1.149436	-0.002156
1	0.89521	0.941481	-0.046271
2	1.40170	1.383238	0.018462
3	0.34094	0.397724	-0.056784
4	1.10691	1.095944	0.010966
...	...	...	...
95	0.85095	0.951171	-0.100221
96	1.42990	1.464582	-0.034682
97	1.48658	1.502701	-0.016121
98	1.16313	1.265230	-0.102100
99	0.55166	0.493125	0.058535

100 rows × 3 columns

Tabla 4.5 Comparativa de resultados para el reactivo biomasa en COBRApy y en el modelo de Red Neuronal EM20797921.

5 OPTIMIZACIÓN DE FLUJO METABÓLICO A PARTIR DE ALGORITMOS GENÉTICOS

En este capítulo se presenta la metodología implementada en el estudio de Algoritmos Genéticos (AG) en la predicción flujo metabólico celular y la viabilidad de su integración al simulador GRO. Se describe el desarrollo durante el proceso de codificación de individuos, cruzamiento, mutación y la determinación de la función de evaluación y objetivo. Además, se presenta un análisis de los resultados que mejor se han adaptado al problema.

5.1 Codificación del problema

Los Algoritmos Genéticos (AG) precisan de una codificación de los parámetros del problema a resolver. La codificación es un aspecto importante ya que constituye la representación de la información y, además, la forma en que son manipulados los parámetros durante el proceso de selección genética. Por lo tanto, la codificación del problema permitirá que el AG tenga la capacidad de converger durante el proceso de selección de individuos.

Para determinar la codificación de los individuos primeramente analizamos la forma de los datos y cómo se representan de forma matemática. En un principio se plantea la resolución de un problema de ecuaciones estequiométricas, que han sido producto de la representación matemática de reacciones químicas para obtener ciertas cantidades de reactivos. Estas ecuaciones pueden representar una reacción entre dos moléculas o todo un conjunto de reacciones como el metabolismo en un microorganismo. En el caso del estudio de microorganismos, se cuenta con gran cantidad de moléculas y reacciones que intervienen en el metabolismo celular, resultando compleja su representación matemática.

En el caso de estudio de esta investigación se utiliza el modelo metabólico *E. coli* K-12 MG1655 que contiene 72 metabolitos y 95 reacciones. Este modelo se encuentra soportado en la herramienta COBRApy para el Análisis de Balance de Flujo Metabólico (FBA) de dicha bacteria, así como la integración de modelos diseñados por el usuario.

El cálculo de FBA en COBRApy se realiza mediante la sentencia *model.optimize()* la cual es la encargada de obtener la función objetivo (biomasa por defecto) mediante las concentraciones de los reactivos. En la Figura 5.1 se observa el resultado de la ejecución de la sentencia anterior, mostrando la cantidad de concentración (fluxes) de cada reactivo y el valor de la función objetivo. En particular se tiene la leyenda "**Optimal solution with objective value 0.874**" que corresponde al valor de producción de biomasa o función objetivo igual a 0.874. La columna fluxes de la Figura 5.1 representa la cantidad de concentración de cada reactivo representada con números enteros o racionales.

La presencia de números racionales en el problema estudiado implica una alta cantidad de valores a estudiar, resultando un problema difícil de codificar en forma binaria. Sin embargo, es posible estudiar el modelo *E. coli* en COBRApy utilizando valores enteros para representar la concentración de un reactivo, sabiendo que a ciertas cantidades de concentración de un reactivo la cantidad de biomasa puede llegar a punto máximo de producción.

```
In [51]: #---Funcion de resolucion FBA---#
         model.optimize()
```

Out[51]: ***Optimal*** **solution with objective value 0.874**

	fluxes	reduced_costs
ACALD	0.000000	6.938894e-18
ACALDt	0.000000	0.000000e+00
ACKr	0.000000	1.040834e-17
ACONTa	6.007250	0.000000e+00
ACONTb	6.007250	1.387779e-17
...	...	...
TALA	1.496984	-1.387779e-17
THD2	0.000000	-2.546243e-03
TKT1	1.496984	-1.387779e-17
TKT2	1.181498	1.387779e-17
TPI	7.477382	-6.938894e-18

95 rows × 2 columns

Figura 5.1 Salida de la sentencia *model.optimize()* en COBRApy.

La representación del modelo metabólico de *E. coli* requiere de referenciar el modelo estequiométrico de la bacteria a un conjunto de parámetros que puedan implementarse en un Algoritmo Genético. Los parámetros esenciales para considerar es la composición de 72 metabolitos y 95 reactivos que componen el modelo estudiado. Al hablar de esta cantidad de parámetros conlleva a desarrollar un problema de optimización exhaustivo y con alto procesamiento computacional. Sin embargo, este trabajo de investigación se centra en optimizar y buscar nuevas herramientas a las ya conocidas.

Para fundamentar la eficiente integración de Algoritmos Genéticos en el simulador GRO, las pruebas desarrolladas deben de minimizar en un principio los valores predicción de FBA en COBRApy. Respecto al tiempo de predicción en COBRApy, se ha obtenido en la Sección 4.3 de este trabajo un tiempo igual a 0.001794 segundos.

Para fines prácticos de estudio y para determinar un valor inicial de partida, se eligió analizar los metabolitos; Dióxido de Carbono [CO_2], Oxigeno [O_2] y Acetato [AC] durante la producción de biomasa. Estos metabolitos son analizados con el fin de conocer su contribución a la producción de biomasa y optimizar los reactivos mediante Algoritmos Genéticos.

Este estudio permitirá, entonces, obtener el tiempo de ejecución de un Algoritmo Genético para optimizar productos metabólicos, sirviendo como herramienta complementaria al simulador GRO.

Además, presenta una comparativa respecto al costo computacional de la herramienta COBRApy, sabiendo que al obtener un tiempo menor de procesamiento para calcular el flujo metabólico este método podría descartarse.

5.2 Parámetros del individuo

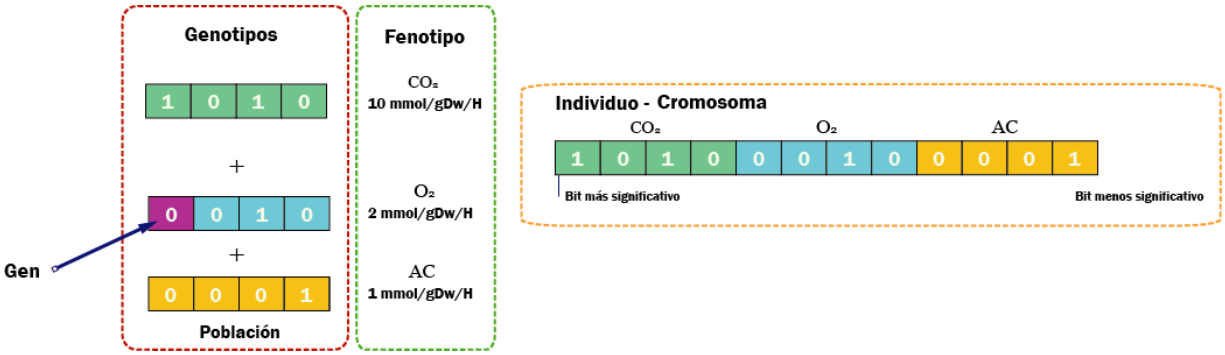


Figura 5.2 Representación y composición de individuos en Algoritmos Genéticos.

Representar la cantidad de nutriente de cada metabolito podrá representarse por un total de 4 bits ya que su conversión decimal permite valores de 0 a 15. De este modo cada nivel de concentración por metabolito estará representada por 4 bits, dando como resultado una composición de 12 bits para conformar un individuo. De forma gráfica se presenta la Figura 5.2 que describe la composición de cada individuo. Es importante mencionar que durante el algoritmo computacional se utiliza un arreglo con capacidad de 12 valores y no un arreglo para cada uno de los metabolitos. Como se observa en la Figura 5.2 el individuo está representado por el conjunto de los 3 metabolitos donde cada 4 bits representan el valor de concentración. Este último parámetro se codifica de forma binara y se integra en su posición dentro del conjunto. Como ejemplo, se encuentra el metabolito O_2 compuesto por 4 genes (bits) con un valor de concentración de 2 mmol/gDw/H (fenotipo) donde al convertir en número 2_{10} a binario resulta el número 0010_2. Lo mismo sucede con los otros 2 metabolitos representando el bit más significativo al metabolito CO_2, posteriormente O_2 y por último el bit menos significativo el metabolito AC.

El proceso de selección sigue el mismo principio mencionado y será descrito en las siguientes secciones.

5.3 Diseño de Población

La población corresponde al conjunto de individuos (cromosomas) que, a través de la reproducción genética, generan soluciones al problema. Este parámetro resulta importante gracias a que dicta la capacidad de búsqueda del algoritmo para llegar a la solución óptima. Además, es un gran factor para determinar el tiempo que lleva al algoritmo en converger. Otro rasgo de la población es que permite evaluar cierta cantidad delimitada dentro de un extenso conjunto de soluciones, evaluando y seleccionando los mejores individuos de cada generación.

Será preciso mencionar que nuestro caso de estudio contiene un total de 4096 (2^{12}) posibles soluciones al problema, hablando de evaluar todos los posibles eventos al presentar los 3 metabolitos en conjuntos de 4 bits cada uno. Acorde con lo anterior, se dispuso a representar el 1% del total de posibles soluciones siendo un aproximado a 40 individuos por población. Esta población permitirá evaluar un pequeño conjunto de la población en cada generación, a la vez que se evalúa y se obtienen los mejores individuos de cada generación.

# Individuo	CO2	O2	AC
1	0011	0010	0011
2	0000	0111	0110
3	0011	0110	0111
4	0010	0010	0011
5	0010	0011	0110
6	0100	0001	0011
7	0100	0111	0000
8	0110	0000	0110
9	0110	0000	0101
10	0111	0110	0011

Figura 5.3 Representación de 10 individuos de la población.

Los individuos de la población son creados de forma aleatoria con valores enteros de entre 0 a 15 (4 bits por cromosoma) lo que permite evaluar hasta 40 individuos por generación antes del proceso de cruza y mutación. La Figura 5.3 muestra la representación de 10 de 40 ejemplares de una primera generación. De esta forma, cada individuo de la generación será evaluador respecto a la función de ajuste y evaluación. El individuo que mejor se adapte al problema será etiquetado como "mejor" y pasará a la siguiente generación.

5.4 Definición de la Función de restricción

La singularidad de un problema a resolverse por Algoritmos Genéticos está determinada por las funciones de evaluación y restricción. Esta última es la encargada de evaluar el grado de adaptación de los individuos, por lo que su determinación, hace posible evaluar la efectividad de los individuos respecto a la solución óptima.

La idea general de la aplicación de Algoritmos Genéticos en predicción del flujo metabólico es poder optimizar el tiempo de procesamiento de un modelo metabólico. Una referencia de estudio actual es la herramienta COBRApy que es capaz de realizar este cálculo de forma automática. Su forma de calcular el crecimiento celular es mediante una función objetivo donde el requerimiento preciso se refiere a la producción de biomasa. Calcular este producto se realiza mediante las reacciones estequiométricas del modelo celular y calculando su concentración respecto a las

cantidades de nutrientes en cada reactivo. Es sabido que la cantidad de concentración de los reactivos determina la cantidad de producción de productos y biomasa; entonces, la ausencia o falta de concentración de un reactivo afecta a gran escala los productos finales.

Un aspecto importante por investigar es poder maximizar la cantidad de productos durante el metabolismo de una célula, minimizando la cantidad de nutrientes en los reactivos. De esta forma sería posible conocer las cantidades mínimas para crear cierta cantidad de reactivos, teniendo cierto control de crecimiento celular. Estas cantidades son difíciles de conocer por medio del modelo estequiométrico de la célula. La razón es que ciertos reactivos contribuyen a mayor o menor escala en la producción de ciertos productos. Conocer los principales reactivos que limitan la producción de biomasa resulta un punto importante para estudiar y como objeto de investigación.

La forma que tenemos para medir el grado de adaptación de los individuos es por medio de la cantidad de biomasa producida a cierta cantidad de concentraciones de los reactivos. Obtener la biomasa se logra por medio de las ecuaciones estequiométricas y en este punto se hace conveniente utilizar la herramienta COBRApy por medio de la sentencia *model.optimize()* (vista también en la Sección 4.2).

Función restricción: model.optimize()

Medir el nivel de adaptación de los individuos de cada generación estará dada por la cantidad del producto de biomasa, a mayor cantidad de producto mejor se considerará al individuo como óptimo a la solución. Con base al experimento, se evaluarán las cantidades de concentración de los reactivos 'CO_2', 'O_2' y 'AC' en la producción de biomasa, con la finalidad de encontrar la cantidad mínima de concentración de los reactivos para producir un máximo de biomasa.

5.5 Función de fitness (evaluación)

La función de evaluación será la encargada de evaluar el costo de cada individuo respecto a otros. También corresponde a la función que dicta la penalización o coste del resultado de cada individuo.

Como se ha dicho anteriormente, se intenta buscar la mínima concentración de los reactivos para obtener un máximo de producto (biomasa). Por tal motivo, la cantidad de nutrientes de cada individuo es la razón que se busca penalizar (minimizar). La forma de penalizar a los individuos estará dada por la suma de la cantidad de concentración de los reactivos menos el valor absoluto de la diferencia de dos de los reactivos menos influyentes (menor concentración) al reactivo con mayor concentración. Por lo tanto, la función de evaluación estará dada por la expresión:

$$f_{evaluación} = (m_{CO2} + m_{O2} + m_{AC}) - abs(m_{CO2} - m_{O_2} - m_{AC})$$

donde *m*: es la cantidad de concentración por reactivo.

Función de evaluación (cuando CO_2 presenta la mayor concentración de los reactivos).

De esta forma, el valor de la función de evaluación indicará el costo de producción de biomasa para cierto individuo, sabiendo que a menor valor representará un mínimo de concentración de los reactivos.

5.6 Fase Reproductiva

5.6.1 Selección

La reproducción es el proceso mediante el cual los individuos se reproducen para generar nuevos descendientes. Concretamente la reproducción consiste en seleccionar los mejores candidatos de una población cumpliendo la función de preservar a los mejores individuos hacia las futuras generaciones.

La forma de selección de los mejores individuos de la generación entre los tres reactivos analizados constará del método conocido como torneo. Este tipo de selección consiste en seleccionar de forma aleatoria un número de individuos pertenecientes a la generación para someterlos a una evaluación o torneo. De los individuos seleccionados, aquellos que presenten la menor penalización, tras desarrollar la función de evaluación, son considerados como ganadores y posteriormente conforman a los individuos padres para el proceso de cruzamiento.

5.6.2 Cruce

El proceso de cruce cumple la función de preservar el material genético de los padres hacia la siguiente generación capaz de copiar parte del material genético de los padres y crear un nuevo individuo a partir de ellos.

El tipo de cruce a implementar entre los reactivos (metabolitos) “padre” corresponde al cruce de un solo punto. Esta metodología toma un punto aleatorio de un cromosoma padre y copia la cadena genética desde el inicio hasta el punto de cruce y el resto del material genético es copiado del otro padre. De esta forma se obtienen dos nuevos descendientes que heredan material genético de sus padres y que formaran parte de la siguiente generación.

Para implementar esta metodología de cruce se elige aleatoriamente un punto *P* para ser cortados los cromosomas en ambos puntos. Mediante codificación binaria se realiza copiando los bits pertenecientes al punto de cruce de un individuo y posteriormente se complementa con el extremo faltante del otro individuo. Para ilustrar mejor se presentan las imágenes de las Figuras 5.4 y 5.5, las cuales presentan puntos de corte diferentes y “unión diversa” entre segmentos.

Procedimiento para método de cruza binario

- Inicialmente se elige aleatoriamente un punto ***P*** dentro del rango de dimensiones del arreglo binario que representa a los individuos. En la Figura 5.4 corresponde al punto **P** igual a 2 por lo que cada arreglo binario que representa a los reactivos se corta en el segundo bit de izquierda a derecha. Cabe señalar que no importa si se corta el arreglo de izquierda a derecha o de derecha a izquierda siempre y cuando se preserve la conformidad en el proceso de cruza.

- kSe seleccionan los individuos 1 y 2 para realizar el respectivo corte. Referente al individuo 1 se toma la parte izquierda del arreglo para ser copiada al primer descendiente. Posteriormente elegimos el complemento del individuo descendiente respecto al complemento del punto **P** de cualquier segmento del individuo 2. Se observa en la Figura 5.4 que el primer segmento del primer descendiente corresponde al 2o bit de CO_2 del individuo 1 y 2o bits de AC del individuo 2. Como se observa, en el punto de corte se toma la parte izquierda del punto P del individuo 1 y el segmento derecho del individuo 2.

- Nuevamente se toma el segmento izquierdo de otro reactivo del individuo 1, correspondiente a la Figura 5.4 se toma la parte izquierda del reactivo O_2 y se complementa con el segmento derecho de CO_2 del individuo 2.

- Por último, copiamos la parte izquierda del punto **P** del reactivo AC y lo complementamos con el segmento derecho del reactivo O_2 del individuo 2. Los segmentos correspondientes se integran para conformar un individuo descendiente.

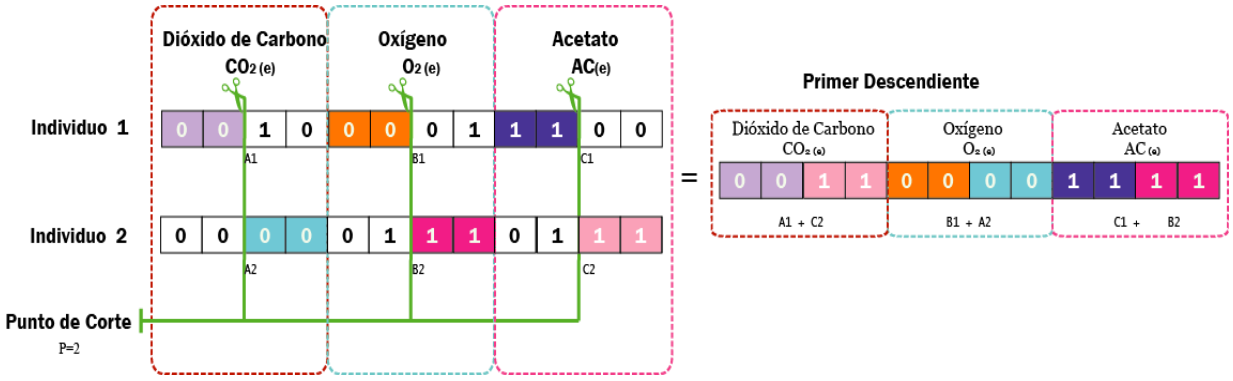


Figura 5.4 Proceso de cruza entre individuos con punto de cruza igual a 2.

Para garantizar mayor diversidad de los individuos, se utiliza un nuevo punto de corte para crear los siguientes descendientes.

A modo de ejemplo se presenta la Figura 5.5 con un punto de corte igual a tres. En este ejemplo se presenta otro conjunto de individuos “padre” para la descendencia. Inicialmente se cortan todos arreglos binarios que representan a los reactivos en el punto 3. Se toma como primer segmento el

componente izquierdo de CO_2 del individuo 2 y se complementa con el segmento derecho de AC del individuo 1. Siguiendo los pasos, se copia el segmento izquierdo al punto **P** de O_2 del individuo 2 y se complementa con el segmento derecho de O_2 del individuo 1. Por último, se copia el extremo izquierdo de AC del individuo 1 y se complementa con el segmento derecho de CO_2 del individuo 1. De esta forma se crea el nuevo descendiente entre ambos individuos "padre".

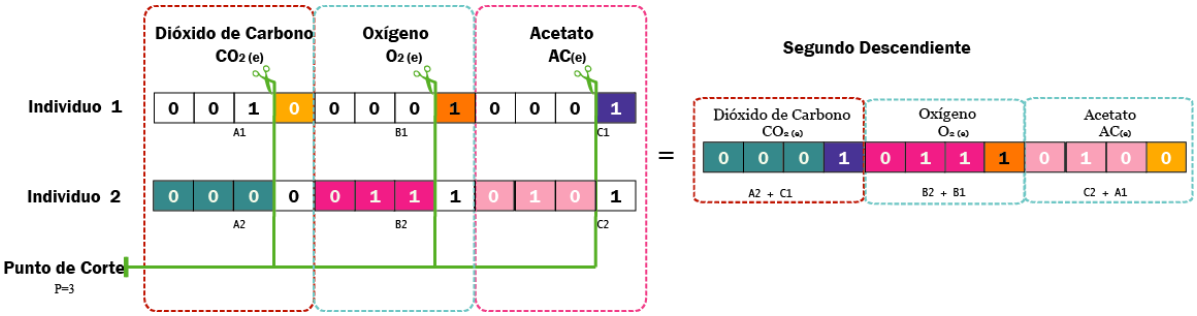


Figura 5.5 Proceso de cruza entre individuos con punto de cruza igual a 3.

Los ejemplos anteriores se apoyan como muestra a la forma de cruza de los individuos para garantizar la diversidad hacia las futuras generaciones. Acorde con ello, se establecen diferentes secciones de cruza entre los diferentes reactivos de los individuos.

5.6.3 Mutación

La mutación provee la diversidad genética de los individuos permitiendo cubrir y evaluar un mayor número de individuos genéticamente diferentes. En el caso de los Algoritmos Genéticos se lleva a cabo mediante el cambio aleatorio de algunos de los genes de los cromosomas, considerando la probabilidad de cruce establecida.

Desarrollar la mutación para el caso de estudio, consiste en intercambiar de forma aleatoria el valor de una posición del individuo. Así, por ejemplo, si en un punto aleatorio **M** del individuo el valor del bit es un 0 su valor será cambiado a 1. Este proceso es desarrollado con un individuo aleatorio de la generación y repetido con una probabilidad del 2% sobre cada generación. Con estos parámetros garantizamos diversidad de los individuos y se evita que la mutación se aplique generación tras generación evitando mutaciones importantes en cada generación que alteren la descendencia óptima y garantizando diversidad genética.

Conforme al proceso de mutación que se ha implementado, se muestra la Figura 5.6. Se puede observar que para un punto aleatorio **M** igual a 5, se selecciona el quinto bit (desde el menos significativo) con un valor igual a 0. Posteriormente y respecto al valor de bit se cambia el valor a su contrario. Por lo tanto, en el individuo después de la mutación el quinto bit presenta el valor de 1.

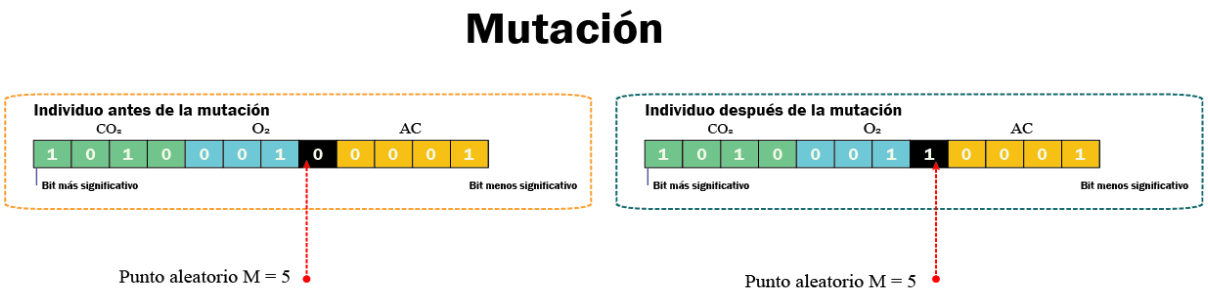


Figura 5.6 Proceso de mutación para M = 5

5.6.4 Condición de término

El principal objetivo del Algoritmo Genético es encontrar un óptimo local o general al problema dado. Lograr lo anterior conlleva a repetir cierto número de veces los procesos de la fase de reproducción. Sin embargo, el Algoritmo Genético debe contar con una condición de término para evitar llegar a un ciclo infinito. Una forma de condicionar al algoritmo a detenerse es haber alcanzado el óptimo global o hasta llegar a la evaluación de cierta cantidad de generaciones. Esta última opción parece ser la más viable de implementar ya que nuestro caso de estudio contiene una gran cantidad de soluciones que satisfacen al problema y, como se ha explicado en la Sección 5.1, el problema ha sido delimitado para un objeto de estudio.

La manera en que se decide delimitar el algoritmo es considerando a los mejores individuos producto de la evolución de 100 generaciones. Este parámetro es considerado en un principio para evaluar el tiempo de ejecución del algoritmo genético con tres reactivos del modelo metabólico estudiado. De esta forma obtendremos un valor que permitirá conocer el tiempo mínimo para encontrar los mínimos valores de concentración para maximizar la biomasa. Es importante porque así conocemos si es factible su implementación en el simulador GRO. Lo que se espera encontrar después de su implementación es tener valores de tiempo inferiores a 1 minuto, puesto que es sabido que durante una simulación de colonias de microorganismos se procesan cientos de cálculos metabólicos, haciendo lentas las simulaciones en tiempo real. En otros términos, se puede considerar su implementación en el simulador como una herramienta externa a la simulación en tiempo real y más como una herramienta que ayuda al usuario a conocer los límites de concentración que debe brindar para garantizar el crecimiento de las células que se estudien.

5.7 Convergencia del algoritmo

Los individuos que mejor se adaptan al problema planteado han de ser aquellos que presentaron los mejores genes tras el paso de las generaciones. La forma en que logramos que el algoritmo implementado llegue a encontrar los valores óptimos es mediante la repetición de los procesos de

selección, cruza y mutación. En un inicio se ha de crear la primera generación de 40 individuos, los cuales son evaluados con la función de restricción y evaluación, con el fin de encontrar el mejor individuo de la generación. La descendencia genética a futuras generaciones se lleva a cabo mediante el intercambio genético de dos individuos "padres" que heredan parte de su contenido genético al descendiente. Aunado a ello, la mutación es desarrollada para garantizar la diversidad de individuos en las generaciones. El Algoritmo Genético busca la solución óptima al problema mediante la evolución de la fase reproductiva y termina cuando se han evaluado 100 generaciones.

5.8 Implementación del algoritmo

5.8.1 Pseudocódigo implementado

Respecto al algoritmo llevado a cabo en el desarrollo experimental, se presentan las Figuras 5.7 y 5.8 referentes al pseudocódigo utilizado. Principalmente se codifica una población inicial con tamaño igual a *dimPoblacion.* Con una primera población se procede a la fase de evolución mediante el ciclo reproductivo. Este ciclo consta de evaluar los individuos de la generación con la función de evaluación y restricción para seleccionar de entre ellos el individuo con mejor adaptación. Posteriormente, se realiza el proceso de cruza mediante la selección de individuos ganadores del torneo hasta completar una nueva generación (el último individuo es el mejor adaptado respecto a las funciones de adaptación y restricción). La diversidad de búsqueda se logra además con el método de mutación con probabilidad sujeta. Aquel individuo mejor adaptado de toda la generación pasa de forma directa a la siguiente generación para completar la nueva población. El proceso evolutivo se repite hasta alcanzar las 100 generaciones dando como resultado el individuo que aporta la mejor solución al problema.

Las Figuras 5.7 y 5.8 tienen la finalidad de mostrar el pseudocódigo base para la creación del algoritmo. El código implementado en lenguaje Python se encuentra en el Apéndice de este trabajo.

Variables de Entrada

prob	/*Probabilidad de mutación */
maxGen	/*número máximo de generaciones */
dimCromosoma	/*Dimensión del cromosoma */
maxConcentracion	/*Concentración máxima por reactivos */
dimPoblacion	/*Dimensión de población */
arrayReactivos	/*Nombres de reactivos */

Variables de Salida

mejorIndividuo	/* Mejor individuo por generación */
tiempoEjecucuion	/* Tiempo de Ejecución de programa*/
maxBiomasa	/* Cantidad máxima de biomasa por individuo */

Figura 5.7 Variables del Algoritmo Genético implementado.

```
Begin algoritmoGenetico
   -Generar población inicial de tamaño dimPoblacion
   -Codificar población inicial
   -Igualar contadorGeneracion a cero
   /* INICIO de ciclo reproductivo */
   While contadorGeneracion sea diferente a  maxGen
      - Evaluar F. evaluación y F. restricción de los individuos de la generación.
      - Seleccionar el mejorIndividuo de la generación
      /* INICIO de generar nueva población */
      For dimPoblacion - 1  Do
         - Seleccionar dos individuos aleatorios de la población para torneo.
         - Cruzar los individuos ganadores del torneo.
         - Agregar los descendientes de la cruza a la nueva generación
         - Mutar un individuo sujeto a probabilidad de prob
      /* FIN de generar nueva población */
      - Agregar mejorIndividuo de la generación
   /* FIN de ciclo reproductivo */
   - Mostrar mejorIndividuo
   - Mostrar maxBiomasa
   - Mostrar tiempoEjecucion
END
```

Figura 5.8 Pseudocódigo de Algoritmo Genético implementado.

5.8.2 Importancia de la mutación

Un aspecto fundamental que estudiamos es la importancia de la correcta aplicación de la mutación, ya que ésta permite la diversidad de los individuos de la población haciendo que las alteraciones en la secuencia genética influyan en los individuos a su adaptación al entorno. Por tal motivo se presentan los experimentos realizados al aplicar un algoritmo con mutación por cambio completo en la secuencia genética y otro aplicando mutación por cambio único de un gen de la secuencia genética.

La Figura 5.9 muestra los resultados de la función de evaluación aplicada al mejor individuo de cada generación, observando una mutación por medio de la alteración de todos los genes del cromosoma, que como se puede observar, aplica una mutación tan grave de los genes que evita la búsqueda eficiente de la solución. Por medio de la gráfica se puede observar que el valor se maximiza en 18 unidades y se mantiene así durante las 50 generaciones haciendo que el algoritmo se encuentre en un punto no óptimo. Esta mutación provoca que todos los genes se vean alterados y hace que los individuos no logren mejorar respecto a futuras generaciones debido a que al alterar todos los genes de un individuo la descendencia se pierda al paso de las generaciones.

En la Figura 5.10 se presenta la proyección de la función de evaluación después de 50 generaciones utilizando la mutación por cambio en un único gen del cromosoma. Por medio de la gráfica de la Figura 5.10 se puede apreciar la evidente función de la mutación: preservar el código genético y otorgar diversidad a la población. Se puede analizar que cambiando el valor de un único gen del cromosoma permite que el algoritmo pueda converger de forma más optima, además de evaluar un mayor número de individuos en la búsqueda de soluciones. Se observa en la Figura 5.10 cómo el valor de la función de evaluación en el mejor individuo de cada generación se minimiza al paso de las generaciones, que para la generación 10 la función de evaluación arroja valores mínimos al paso de las generaciones, observando que le ha tomado al algoritmo de la generación 10 a la 30 reducir a un mínimo, permitiendo al algoritmo converger de forma óptima y presentar mejoras respecto a la mutación por un cambio completo en el código genético.

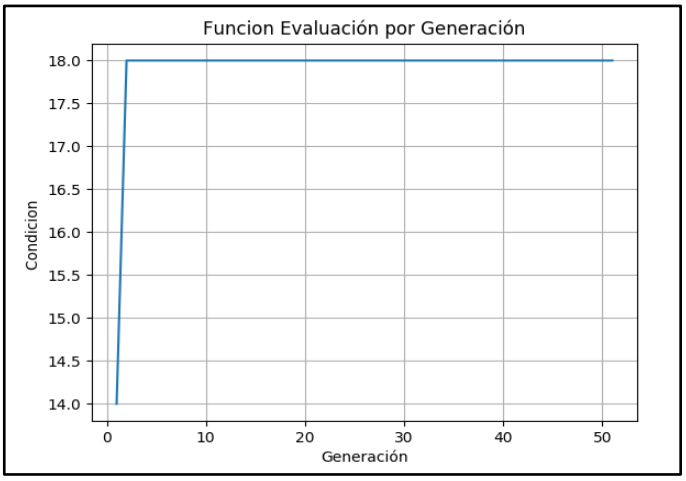


Figura 5.9 Proyección de la función de evaluación tras generación aplicando mutación por cambio total de la secuencia genética.

Con el análisis anterior optamos por aplicar al Algoritmo Genético una mutación por cambio único de un gen de la secuencia genética, presentando mejores resultados al paso de las generaciones.

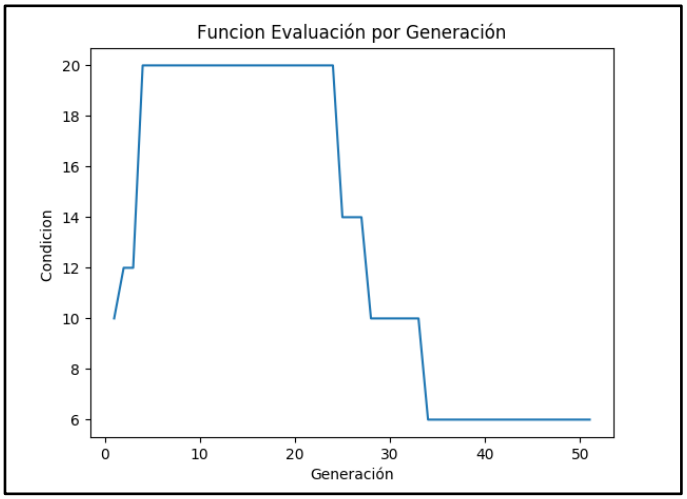


Figura 5.10 Proyección de la función de evaluación tras generación aplicando mutación por cambio de un único gen en la secuencia genética.

5.8.3 Evolución del algoritmo y resultados

A continuación, se presentan los resultados referentes al proceso de evolución del Algoritmo Genético implementado, observando los resultados obtenidos tras 40 generaciones. Los resultados presentes a continuación se han procesado durante 15.904 segundos de ejecución en el entorno Pycharm. Las Figuras 5.11, 5.12 y 5.13 muestran las gráficas analizadas en este experimento.

Los primeros parámetros para analizar refieren a la secuencia de valores que arroja la función de evaluación en el mejor individuo de cada generación y cuál es su influencia de la cantidad de nutrientes respecto a ello. Considerando ahora las proyecciones de las Figuras 5.11 y 5.12 se aprecia cómo durante las 5 primeras generaciones el valor de la función de evaluación esta maximizado en 30 unidades y esto es debido a que los valores de las concentraciones de los reactivos CO_2, O_2, se encuentran en 15,5 y 12 unidades mmol/gDW/H, respectivamente, siendo estas concentraciones altas dentro de las mínimas posibles. Como es sabido, la condición de optimización está destinada a minimizar la concentración de los reactivos para obtener la mayor producción de biomasa por lo que, respecto a la Figura 5.13, se observa cómo el valor de la biomasa durante las primeras 5 generaciones ha alcanzado el punto máximo, haciendo eficiente la decisión de la función de restricción.

Ahora observamos en la Figura 5.11 cómo el valor de la función de evaluación comienza a descender a partir de la quinta generación. Haciendo un análisis de la Figura 5.12 podemos destacar los puntos siguientes:

- El reactivo AC presenta en 0 unidades de concentración en la sexta generación,
- El reactivo CO_2 mantiene una concentración de 15 unidades desde la segunda y cuadragésima generación,
- Durante la sexta y décimo novena generación, el reactivo O_2 disminuye su concentración de 3 a 0 unidades; sin embargo, el reactivo AC aumenta de 0 a 1 unidad,

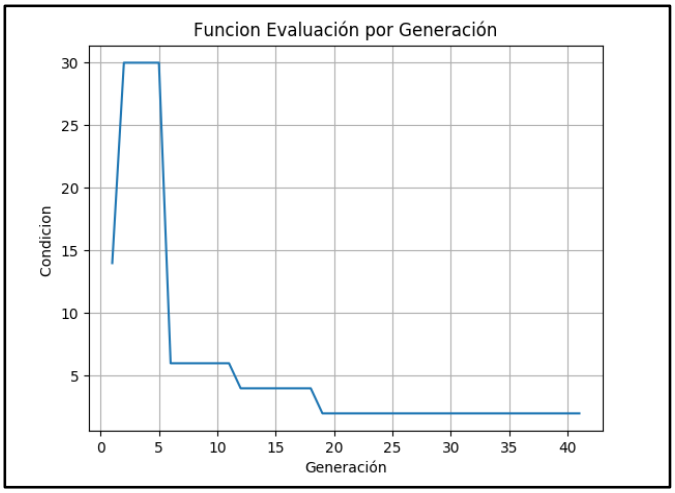


Figura 5.11 Proyección de la función de evaluación tras aplicar al mejor individuo.

Un punto destacable es hablar de lo que sucede en la generación 19, observando que el valor de la función de evaluación se ha minimizado a 2 unidades, siendo este valor el óptimo local de la solución después de 40 generaciones. Se observa que al alcanzar el mínimo local los valores de los reactivos se mantienen estables después de la generación 19, siendo las concentraciones de los reactivos establecidos en 15 unidades en CO_2, 0 unidades para O_2 y 1 unidad para AC. A parir de dichas concentraciones podemos inferir que, para la producción de 0.7667683 unidades de biomasa se necesitan al menos 15 CO_2, 0 O_2 y 1 AC unidades de mmol/gDw/H por reactivo.

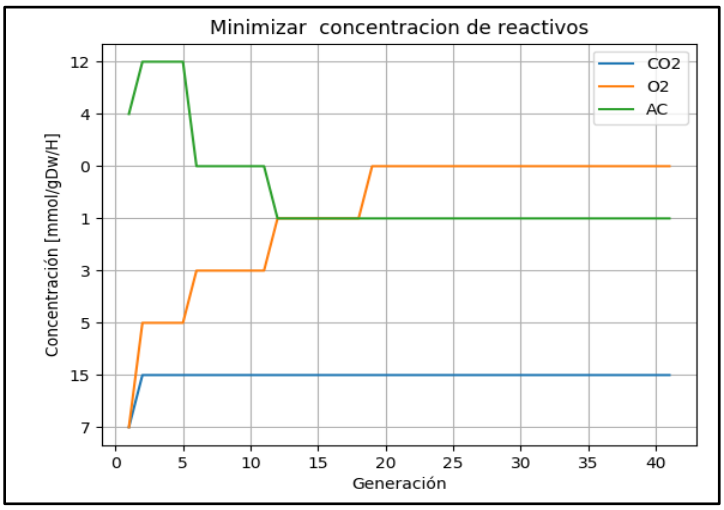


Figura 5.12 Proyección de la cantidad de concentración de los reactivos al paso de las generaciones.

La singularidad que aportan los Algoritmos Genéticos es poder evaluar de forma rápida un pequeño segmento del conjunto de posibles soluciones al problema. Como se ha mencionado anteriormente, esto se logra gracias a las fases de la selección de los mejores individuos de cada población y que dentro del proceso de selección influye significativamente el conjunto perteneciente a la primera generación y sus descendientes. Pongamos por caso la Figura 5.14 que representa la proyección de la función de evaluación para otra ejecución del algoritmo, con una población diferente a la analizada en la Figura 5.11. Se observa cómo en la Figura 5.14 el mínimo local de las 40 generaciones se ha alcanzado en la décima generación, ubicándose en 6 unidades. Ahora vemos la cantidad de nutrientes en la Figura 5.15 y cómo sus cantidades a partir de la décima generación se encuentran en 3 unidades del reactivo AC, 0 unidades del reactivo O2 y 15 unidades del reactivo CO2.

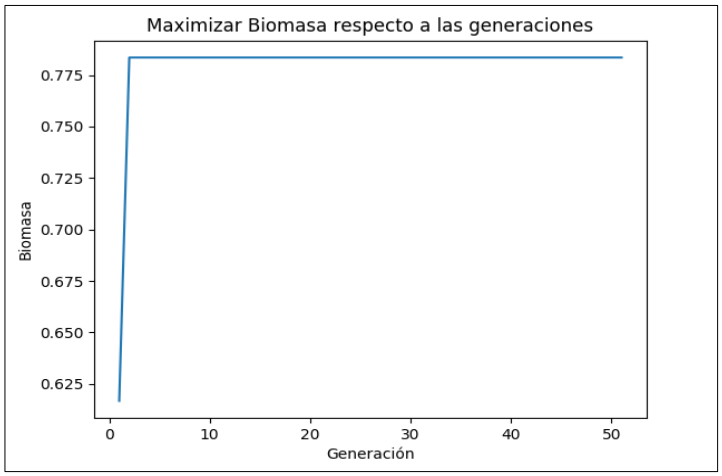


Figura 5.13 Proyección de la función de restricción evaluada en el mejor individuo.

Con lo expuesto en el párrafo anterior, se comprende claramente la singularidad de los Algoritmos Genéticos de evaluar rápidamente un conjunto delimitado de la población y que al utilizar poblaciones aleatorias se obtienen resultados diferentes en cada evento. De manera que en cada evento o ejecución del algoritmo se han de obtener mínimos locales a la solución del problema, cumpliendo la función de optimizar resultados de forma eficiente.

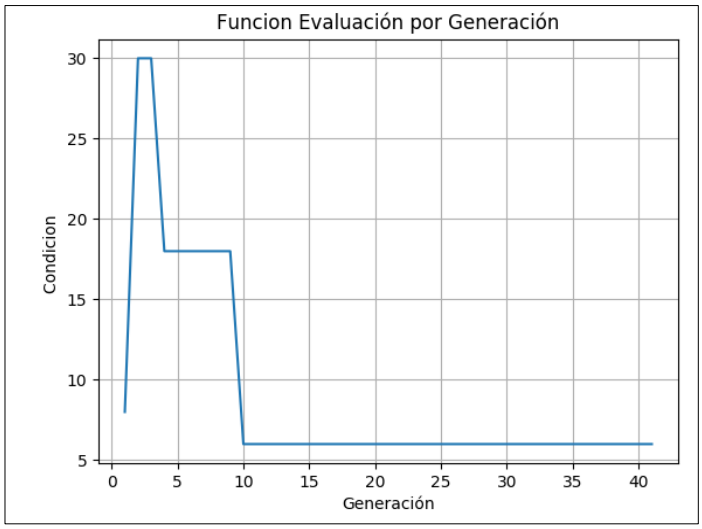


Figura 5.14 Proyección de la función de evaluación para una segunda ejecución del algoritmo.

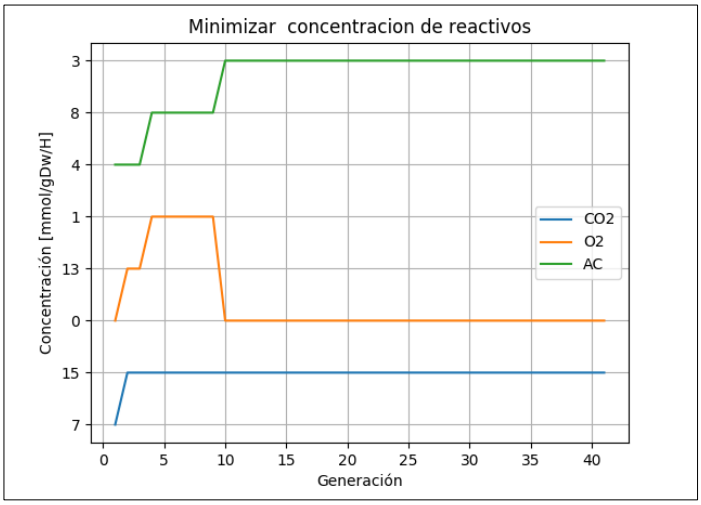


Figura 5.15 Proyección de la cantidad de concentración de los reactivos al paso de las generaciones.

6 INTEGRACIÓN DE RED NEURONAL EN EL SIMULADOR GRO Y TRABAJO FUTURO

Integración de Red Neuronal en Simulador GRO

El principal enfoque del presente capítulo es dar a conocer la forma de integrar el modelo de red neuronal al simulador GRO y su integración como una herramienta complementaria al software. Por lo tanto, se mostrará la forma de extraer el modelo de Red Neuronal hacia otros lenguajes de programación y su posterior ejecución dentro del mismo.

La siguiente descripción se presenta como una metodología para integrar el modelo de red neuronal al simulador GRO o a otras plataformas de desarrollo que precisen del Análisis de Balance de Flujo (FBA) metabólico celular.

Parte de los objetivos de esta investigación es el integrar el modelo de inteligencia artificial al simulador GRO. Sin embargo, por la duración de la investigación y la validación de los resultados obtenidos, se ha documentado esta investigación hasta la comprobación del objetivo principal del trabajo, es decir, reducir los tiempos de predicción de Análisis de Balance de Flujo mediante algoritmos de inteligencia artificial.

Modo de aplicación

La idea principal para integrar los modelos de Red Neuronal es mediante la extracción de los pesos y *bias* que han surgido del entrenamiento en Python.

Principalmente se parte de la arquitectura de capas en la Red Neuronal, así como los parámetros del modelo. En este caso de estudio utilizamos el modelo 'EM2079797921' analizado en la Sección 4.7.2, la Tabla 6.1 resume los principales parámetros de este modelo.

Parámetros del modelo "EM2079797921"

Arquitectura	Datos de entrenamiento	Épocas	Función de activación	Error	Tiempo de predicción
20 – 79 – 79 - 79 – 21	200	500	SELU	0.002741	0.000350507

Tabla 6.1 Parámetros del modelo de implementación "EM2079797921".

Extraer pesos y *bias* del modelo en Keras

Gracias a las herramientas que integra Keras en Redes Neuronales, podemos extraer los pesos y *bias* del modelo entrenado fácilmente. A partir del modelo entrenado, se ejecuta la sentencia:

model.layers[n].get_weights()[p] ; ***n*** *el número de la capa a acceder*
p=0*; pesos*
p =1*; bias*

La sentencia de arriba arroja un arreglo de valores pertenecientes a los pesos de la capa **n.** Los datos contenidos se presentan como un arreglo de valores de *float32 y* se puede acceder a ellos una vez que el modelo se ha definido. Para el uso correcto de la aplicación, se obtendrán los valores de los pesos una vez que el modelo ha sido entrenado con las épocas correspondientes.

A modo de ejemplo se presenta la Figura 6.1, donde se observa la ejecución **In [31]** en el entorno Jupyter destinada a conocer la forma de la cuarta capa de la red, la cual presenta 79 conexiones de entrada y 21 conexiones de salida. Referente a la ejecución **ln [32]** se muestran los valores de los pesos de las 79 neuronas en la cuarta capa oculta de la red (en el entorno Jupyter se muestran a dos cifras después del punto decimal). De la misma manera se puede acceder a los valores correspondientes al *bias* de cada capa. Como sabemos, estos corresponden a un *bias* por neurona respecto a las conexiones de salida de la capa. En la Figura 6.2 se observa la salida de la ejecución **ln [33]** mostrando el *bias* de la primera capa oculta con forma de 21 datos y su correspondiente contenido se visualiza en la ejecución **ln [34].**

```
In [31]: modelFromFile1.layers[4].get_weights()[0].shape

Out[31]: (79, 21)

In [32]: modelFromFile1.layers[1].get_weights()[0]

Out[32]: array([[ 0.25, -0.17, -0.24, ...,  0.26, -0.09, -0.08],
                [-0.22,  0.27, -0.03, ...,  0.15, -0.18, -0.18],
                [ 0.13,  0.06,  0.3 , ..., -0.17,  0.21,  0.24],
                ...,
                [-0.35,  0.11, -0.07, ..., -0.15,  0.01, -0.14],
                [ 0.2 , -0.18,  0.26, ..., -0.06, -0.19, -0.11],
                [-0.12,  0.03,  0.22, ...,  0.02,  0.01, -0.21]], dtype=float32)
```

Figura 6.1 Valores de pesos entrenados en la tercer capa oculta del modelo “EM2079797921”.

```
In [33]: modelFromFile1.layers[4].get_weights()[1].shape

Out[33]: (21,)

In [34]: modelFromFile1.layers[4].get_weights()[1]

Out[34]: array([-0.02,  0.05,  0.05, -0.02,  0.04,  0.01,  0.14, -0.  , -0.13,
                -0.06, -0.03,  0.02,  0.05, -0.04, -0.03,  0.05, -0.1 ,  0.08,
                -0.01,  0.03, -0.04], dtype=float32)
```

Figura 6.2 Valores de bias entrenados en la tercer capa oculta del modelo “EM2079797921”.

Podemos afirmar que para extraer todos los valores del peso y *bias* de cada capa de la red, se deberán ejecutar las sentencias *model.layers[n].get_weights()[n]* según la dimensión de capas de la red a implementar. En general se puede acceder a todos los datos del modelo mediante la sentencia *model.get_weights().*

De forma ordenada y práctica se pueden convertir los pesos y *bias* del modelo a un archivo sencillo de leer como CSV. Este archivo pertenece a un conjunto de datos separado por coma y puede ser leído por un gran número de procesadores de texto y hojas de cálculo, además de presentar soporte para muchos lenguajes de programación.

Una forma de exportar los datos del modelo entrenado es mediante la conversión del arreglo de datos *numpy* a formato *Data Frame.* Este último integra un método para exportar el tipo *DataFrame* a un archivo CSV. Estas sentencias se pueden llevar a cabo mediante las instrucciones de la Figura 6.3 donde se visualiza una parte del código para exportar los datos del modelo.

```
In [123]: #--- Obtener pesos y bias del modelo entrenado ---#
          weightsInputLayer = modelFromFile1.layers[0].get_weights()[0]
          biasInputLayer = modelFromFile1.layers[0].get_weights()[1]


          #--- Convertir array a Data Frame ---#

          dFrameWInputLayer = pd.DataFrame(weightsInputLayer)
          dFrameBInputLayer = pd.DataFrame(biasInputLayer)


          #--- Guardar Data Frame a archivo CSV ---#

          dFrameWInputLayer.to_csv('weightsInputLayer.csv')
          dFrameBInputLayer.to_csv('biasInputLayer.csv')
```

Figura 6.3 Exportar los datos del modelo a un archivo CSV.

El fragmento de código de la Figura 6.3 debe satisfacer a cada una de las capas ocultas de la red, de esta manera se puede tener control de cada uno de los parámetros del modelo. Guardando el *bias* y los pesos en formato CSV es posible leer en un procesador de texto y en un lenguaje de programación como C++ el contenido del archivo exportado.

El simulador GRO esta soportado en lenguaje C++ lo que haría posible la lectura del archivo CSV y poder operar los datos dentro del lenguaje y del entorno simulador. Una vez extraídos los valores del peso y *bias* pertenecientes a cada capa, es posible integrar un código de programa para construir una estructura de Red Neuronal con las capas del modelo estudiado. El código de programa puede ser descrito únicamente por un modelo de red de propagación hacia adelante debido a que los pesos y los *bias* del modelo, ya se encuentran con los valores ajustados, producto del entrenamiento llevado a cabo en el Capítulo 4.

Es preciso señalar que la forma de operar los datos del modelo entrenado en Python hacia C++ (lenguaje soportado en GRO) se realiza por medio de la operación con matrices y vectores, los cuales tienen sus formas precisas de operación y declaración en cada entorno.

En resumen, la metodología a integrar en el simulador es por medio de una herramienta que calcule el flujo metabólico celular a partir de Redes Neuronales entrenadas por separado en Python. Esta herramienta tomaría los pesos y *bias* de entrenamiento en Python mediante archivos CSV. Con los datos cargados a la plataforma sería posible calcular el FBA de modelos metabólicos como *E. coli* u otros modelos de estudio. Para este punto debemos decir que para cada modelo metabólico se debe seguir todo el estudio del capítulo 4 ya que cada modelo metabólico tiene sus propias características y principios, lo que haría que ciertas estructuras de redes neuronales aprendan de diferente manera según su principio estequiométrico.

CONCLUSIÓNES

Este trabajo de investigación aborda el tema de técnicas basadas en Inteligencia Artificial aplicadas al área de la biología de sistemas, cuyo objetivo principal es aportar herramientas y mejoras en los sistemas de simulación de microorganismos a través de su red metabólica. Como parte fundamental se demostró que las Redes Neuronales de tipo retro propagación, representan una técnica de aprendizaje computacional que permite emular el balance de flujo metabólico de forma rápida y con alta precisión, comparada con los modelos hechos con cálculo por ecuaciones estequiométricas.

A partir de los datos analizados, se logró entrenar un modelo de Red Neuronal Artificial capaz de aprender a Analizar el Balance de Flujo (FBA), a través del modelo estequiométrico que representa a la red metabólica de bacterias como *E. coli*. El modelo 'EM2079797921' analizado en la Sección 4.6.3 ha podido demostrar la capacidad de predecir FBA de forma eficiente y es al menos tres veces más rápido respecto a la aproximación por la herramienta COBRApy, ofreciendo mejoras en implementación práctica en simuladores como GRO.

La experimentación con Redes Neuronales nos ha permitido integrar modelos escalables y eficientes al intentar emular redes metabólicas como *E. coli*, aportando una importante herramienta para analizar el flujo metabólico celular de bacterias en ambientes de crecimiento poblacional, siendo el principal enfoque del simulador GRO.

A partir de los resultados obtenidos, se puede destacar la alta probabilidad de que las Redes Neuronales puedan aprender modelos metabólicos más complejos y ser desarrolladas en campos de la ingeniería de sistemas, biología sintética y poder ser llevadas a experimentos de laboratorio tipo *in vitro*.

En lo que respecta a la integración de Algoritmos Genéticos capaces de optimizar la obtención de bio-productos, se demostró que su implementación es viable como una herramienta a simuladores poblacionales con propósitos de conocer los límites de concentración para el crecimiento celular de un agente. Esta herramienta permitirá a los usuarios conocer los reactivos que limitan el crecimiento de la célula y cuáles de ellos contribuyen a mayor escala. Gracias a los parámetros (metabolitos) que representan a los individuos durante el proceso de evolución con Algoritmos Genéticos, permiten a usuarios del simulador saber las concentraciones mínimas de los reactivos que debe haber en el ambiente, permitiendo el desarrollo y crecimiento celular poblacional.

Lo abordado en el Capítulo 5 aporta una significante herramienta cuando se requiere conocer la cantidad de producción o crecimiento celular de cierto agente a cierta cantidad limitada de nutrientes, con fin de obtener un máximo o mínimo de producción de bio-productos. También representa una gran aportación como integración a simuladores celulares, ya que brinda información para conocer los reactivos que están limitando los productos del metabolismo, siendo importante cuando se requiere tener un control del crecimiento celular o durante la producción de cierto compuesto.

Es importante destacar que los experimentos y los resultados obtenidos en este trabajo de investigación están sujetos a los datos experimentales. El modelo de Algoritmos Genéticos implementado y el modelo de Red Neuronal 'EM20767621' representan soluciones locales más que globales y únicas a la solución del problema planteado. Lo anterior implica que ante diferentes datos muestrales y diferente potencia computacional se pueden obtener resultados diferentes; sin embargo, es importante decir que, para nuestros los fines estudiados, los resultados obtenidos satisfacen el objetivo y solución al problema, de manera que aporta las ideas para investigaciones futuras en mejorar aún más los tiempos de predicción de FBA mediante búsquedas exhaustivas de arquitecturas de Redes Neuronales. Además, este trabajo es motor para el estudio de simulaciones poblacionales de bacterias como *E. coli* y *Salmonella*.

Por último, podemos destacar que, a través del desarrollo de la Inteligencia Artificial y el avance de arquitecturas de procesamiento computacional, permitirán obtener mejores resultados en tiempos menores respecto a lo estudiado en este trabajo, lo que hace a esta investigación un punto de partida para el desarrollo de sistemas capaces de emular, predecir y controlar colonias de microorganismos como la microbiota humana.

ANEXO A

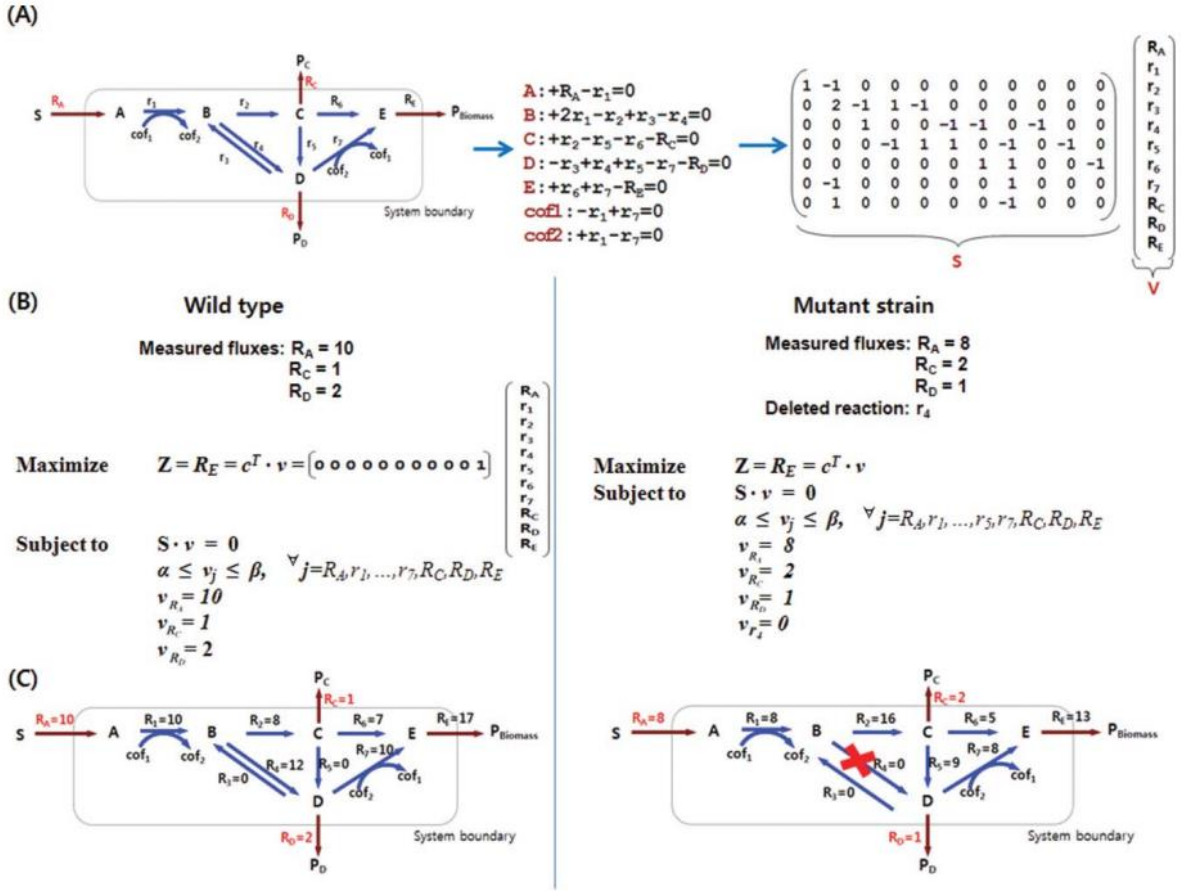


Figura A. Construcción de matriz estequiométrica del modelo de red metabólica y el análisis de flujo basado en restricciones [12].

ANEXO B

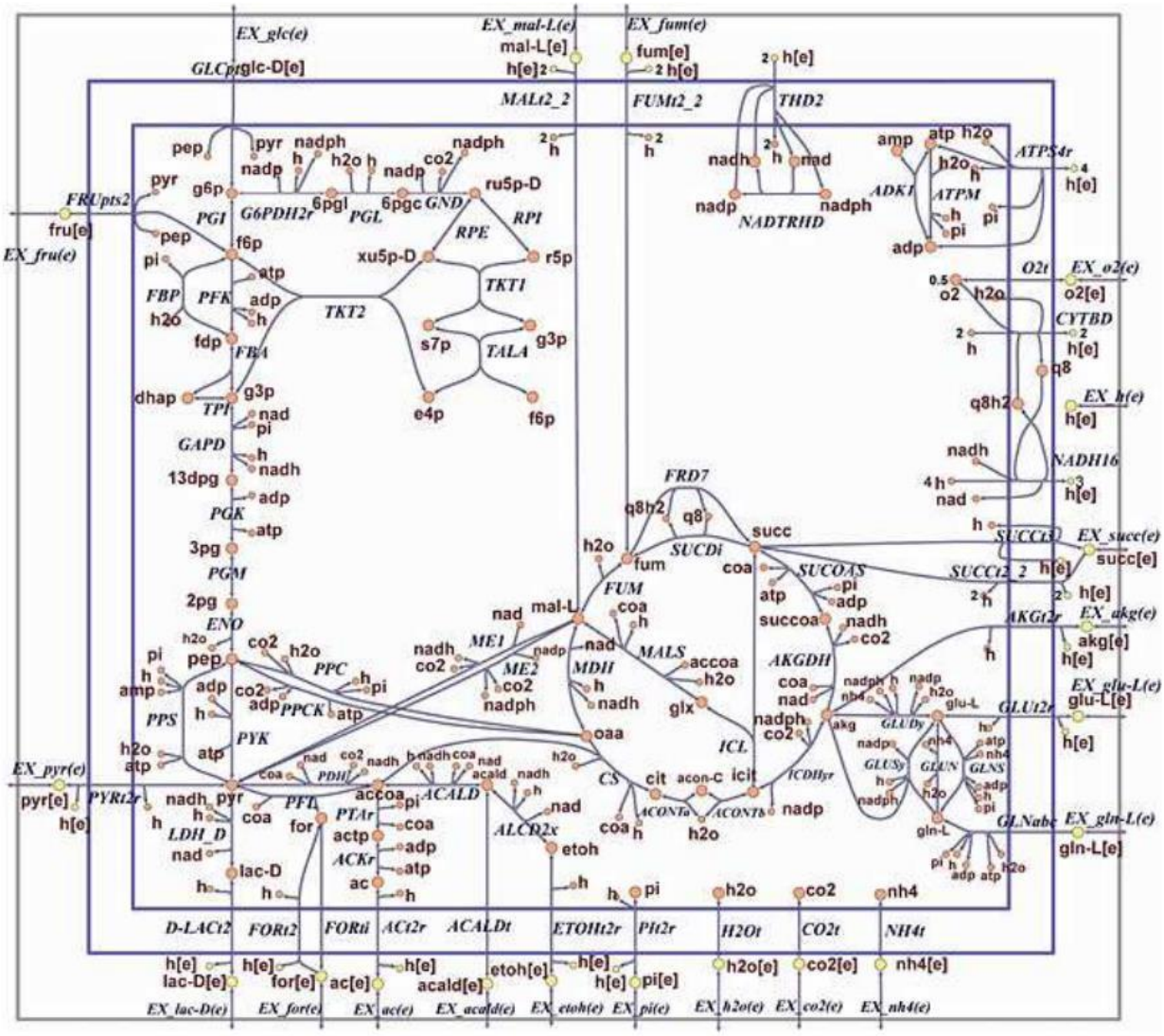


Figura B. Nucleo *E . Coli* K-12 MG1655 Orth, J. D., Palsson , 2014

APÉNDICE

Código de programa referente al Algoritmo Genético implementado en el capítulo 5 para optimizar la producción de biomasa a partir de los índices de concentración de los metabolitos implicados.

```
#---    UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO     -----------------#
#---    FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN  -----------------#
#---    APLICACIÓN DE TÉCNICAS DE INTELIGENCIA ARTIFICIAL -----------#
#---    PARA LA PREDICCIÓN EFICIENTE DEL BALANCE DE FLUJO METABÓLICO-#
#---    CELULAR, Y SU POSTERIOR INTEGRACIÓN EN EL SIMULADOR GRO -----#
#---    INGENIERÍA EN TELECOMUNICACIONES, SISTEMAS Y ELECTRÓNICA ----#
#---    TRABAJO DE TESIS PARA OBTENER EL TÍTULO DE ------------------#
#---    RICARDO JIMENEZ BENITEZ        -----------------------------#


#--- Algoritmo Genetico para la optimizacion de biomasa con ---------#
#--- la mínima cantidad de concentración ----------------------------#

#--------------------------------------------------------------------#

#--- Importar librerías ---#
import numpy as np
import time
import matplotlib.pyplot as plt
from myModelCobra import evaluarModeloCOBRApy


class AlgoritmoGenetico:

    def __init__(self, dimPoblacion=40, maxsol=8, dimGenes=4, maxGeneraciones=40, probMutacion=0.3):

        #--- Variables para codificación ---#
        self.dimPoblacion = dimPoblacion     #--- Dimensión de la poblacion ---#
        self.maxsol = maxsol                 #--- Máximo de limite de concentración ---#
        self.gen = 0
        self.probMutacion = probMutacion     #--- Probabilidad de mutación ---#
        self.poblacion = []                  #--- Individuos de la población---#
        self.dimGenes = dimGenes             #---Dimención de los genes ---#
        self.maxGeneraciones = maxGeneraciones
        #--- Variables para graficar la trascendencia de la optimización y nutrientes ---#
        self.mejorBiomasa = []
        self.arrayCond = []
        self.arrayCO2 = []
        self.arrayAC = []
        self.arrayO2 = []
        #--- Variables para almacenar la biomasa optima y parametros de un individuo ---#
        self.arrayBiomass = []               #--- Contiene el arreglo de la mejor biomasa de cada generacion ---#
        self.mejorIndividuo = [100000000000000, ['0000', '0000', '0000', 0], 0]

```

```
    # Genera la población inicial, según el tamaño de maz sol.
    def generarPrimerPoblacion(self):
        #--- Genera genes aleatorios por metabolito ---#
        dv = np.random.uniform(low=0, high=self.maxsol, size=self.dimPoblacion)    # valor aleatorio limite inferior de metabolito 1
        hv = np.random.uniform(low=0, high=self.maxsol, size=self.dimPoblacion)    # valor aleatorio limite inferior de metabolito 2
        zv = np.random.uniform(low=0, high=self.maxsol, size=self.dimPoblacion)    # valor aleatorio limite inferior de metabolito 2

        #--- Convierte a binarios los genes aleatorios ---#
        print("Población generada: ")
        for i in range(len(dv)):
            #--- Convierte a numeracion binaria ---#
            d = "{0:b}".format(int(dv[i]))
            h = "{0:b}".format(int(hv[i]))
            z = "{0:b}".format(int(zv[i]))

            #--- Completa con ceros segun tamaño del cromosoma ---#
            d = d.zfill(self.dimGenes)
            h = h.zfill(self.dimGenes)
            z = z.zfill(self.dimGenes)

            #--- Visualiza el nuevo individuo generado ---#
            print("  CO2  O2   AC ")
            print(str(i), ":", str(d), str(h), str(z))

            # ---Adiciona nuevo individuo a la polación ---#
            self.poblacion.append([d, h, z, i])


    #---- FASE DE REPRODUCCIÓN ---#
    def reproduccion(self):
        self.data = []
        while True:
            for individuo in self.poblacion:
                self.mejorBiomasa.append(0)
                #--- Evalúa Función Objetivo y Aptitud en cada invididuo ---#
                valorBiomasa = self.evaluarFobjetivo(individuo)
                valorAptitud = self.evaluarFaptitud(individuo)
                #--- Evalúa al mejor individuo ---#
                #--- comparando el valor de individuo ---#
                #--- actual respecto al mejor ---#
                if valorBiomasa >= 0:
                    if valorBiomasa > self.mejorIndividuo[2]:
                        self.mejorIndividuo = [valorAptitud, individuo, valorBiomasa]
                    else:
                        if valorBiomasa == self.mejorIndividuo[2] and valorAptitud < self.mejorIndividuo[0]:
                            self.mejorIndividuo = [valorAptitud, individuo, valorBiomasa]
                #--- Evalúa la condicion de termino de reproducción ---#
                # --- mediante las maximas generaciones permitidas ---#
                if not self.maxGeneraciones:
                    if valorBiomasa > 0:
                        return individuo
                else:
                    if self.gen > self.maxGeneraciones:
                        return self.mejorIndividuo

            #--- Crea nueva generación mediante la
            #--- fase de selccción ---#
            #--- SELECCIÓN --#

            nuevosDescendientes = []   #--- Almacena nuevos descendientes  ---#
            count = 0                  #--- Contador de la nueva generación ---#

```

```
107             #--- FASE DE TORNEO ALEATORIO ---#
108             i = 0
109             while i < ((len(self.poblacion)) / 4):
110                 #--- Selecciona dos contrincantes al azar ---#
111                 contrincantes = np.random.uniform(0, len(self.poblacion), size=2)
112                 primerContrincante = int(contrincantes[0])
113                 segundoContrincante = int(contrincantes[1])
114
115                 #--- Primer torneo ---#
116                 #--- compara los valores de las ---#
117                 # ---funciones de aptitud de los contrincantes ---#
118                 if self.evaluarFaptitud(self.poblacion[primerContrincante]) < self.evaluarFaptitud(self.poblacion[segundoContrincante]):
119                     ganadorUno = primerContrincante
120                 else:
121                     ganadorUno = segundoContrincante
122
123                 #--- Segundo torneo ---#
124                 contrincantes = np.random.uniform(0, len(self.poblacion), size=2)
125                 primerContrincante = int(contrincantes[0])
126                 segundoContrincante = int(contrincantes[1])
127                 if self.evaluarFaptitud(self.poblacion[primerContrincante]) < self.evaluarFaptitud(self.poblacion[segundoContrincante]):
128                     ganadorDos = primerContrincante
129                 else:
130                     ganadorDos = segundoContrincante
131
132                 #--- FASE DE CRUZA ---#
133                 descendiente = self.cruzarIndividuos(self.poblacion[ganadorUno], self.poblacion[ganadorDos])
134                 descendiente[0].append(count)
135                 count += 1
136                 descendiente[1].append(count)
137                 count += 1
138
139                 #--- Añadir nuevos individuos a la población ---#
140                 nuevosDescendientes.append(descendiente[0])
141                 nuevosDescendientes.append(descendiente[1])
142                 i += 1 #--- contador de control del torneo ---#
143
144
145             #--- FASE MUTACIÓN ---#
146             i = 0   #--- contador de control para mutación ---#
147             while count < len(self.poblacion) - 1:
148                 #--- Selección de individuo al azar para mutar ---#
149                 numIndividuo = np.random.randint(0, len(self.poblacion))
150                 individuoMutado = self.mutarIndividuos(self.poblacion[numIndividuo][:3])
151                 individuoMutado.append(count)
152                 nuevosDescendientes.append(individuoMutado)
153                 count += 1
154                 i += 1
155
156             #--- Asigna la población resultante de la fase de selección
157             #--- como una nueva generación---#
158             self.poblacion = nuevosDescendientes
159
160             # Agregamos el mejor de la generación a la próxima
161             best = self.mejorIndividuo[1]
162             best.append(count)
163             self.poblacion.append(best)
164             self.gen += 1
165             self.data.append([self.gen, self.mejorIndividuo])
166
167             #--- Visualiza mejor individuo de la generación ---#
168             print("Mejor solución de la generación: ", str(self.mejorIndividuo))
169             print("CO2:", str(int(self.mejorIndividuo[1][0], 2)))
170             print("O2:", str(int(self.mejorIndividuo[1][1], 2)))
171             print("AC:", str(int(self.mejorIndividuo[1][2], 2)))
172             print("Biomasa:", self.evaluarFobjetivo(best))
173
174             #--- Guarda los datos: Biomasa y nutrientes por metabolito ----#
175             #--- con el fin de mostrar graficas al respecto ---------------#
176             self.arrayBiomass.append(self.evaluarFobjetivo(best))
177             self.arrayCond.append(self.mejorIndividuo[0])
178             self.arrayCO2.append(str(int(self.mejorIndividuo[1][0], 2)))
179             self.arrayO2.append(str(int(self.mejorIndividuo[1][1], 2)))
180             self.arrayAC.append(str(int(self.mejorIndividuo[1][2], 2)))
```

```
181
182     #--- Función Aptitud ---#
183     def evaluarFaptitud(self, cromosoma):
184         #--- Secciona el cromosoma para cada metabolito ---#
185         d = float(int(cromosoma[0], 2))
186         h = float(int(cromosoma[1], 2))
187         z = float(int(cromosoma[2], 2))
188         #--- Evalúa el costo del individuo ---#
189         costo = d + h + z
190         #--- Evalua la función de aptitud ---#
191         fAptitud = abs ( abs(d - h - z) - (costo) )
192         return fAptitud
193
194     #--- Función Objetivo --- #
195     def evaluarFobjetivo(self, cromosoma):
196         #--- Secciona el cromosoma para cada metabolito ---#
197         d = float(int(cromosoma[0], 2))
198         h = float(int(cromosoma[1], 2))
199         z = float(int(cromosoma[2], 2))
200         #--- Evalúa la función de aptitud ---#
201         fObjetivo = round(evaluarModeloCOBRApy(d,h,z),8)
202         return fObjetivo
203
204     #--- Proceso de cruza ---#
205     def cruzarIndividuos(self, cromosoma1, cromosoma2):
206         #--- Segmenta los cromosomas por metabolitos ---#
207         a1 = cromosoma1[0]
208         b1 = cromosoma1[1]
209         c1 = cromosoma1[2]
210
211         a2 = cromosoma2[0]
212         b2 = cromosoma2[1]
213         c2 = cromosoma2[2]
214
215         #--- Genera dos descendientes a través de dos cromosomas progenitores ---#
216         #--- Genera un punto P aleatorio --#
217         pCruza = np.random.randint(2, 7)
218         na1 = a1[:pCruza] + b2[pCruza:]
219         nb1 = b1[:pCruza] + a2[pCruza:]
220         nc1 = c1[:pCruza] + b2[pCruza:]
221
222         #--- Genera un punto P aleatorio --#
223         pCruza = np.random.randint(2, 7)
224         na2 = a2[:pCruza] + b1[pCruza:]
225         nb2 = b2[:pCruza] + a1[pCruza:]
226         nc2 = c2[:pCruza] + b1[pCruza:]
227         #--- Regresa individuos producto de la cruza ---#
228         return [[na1, nb1, nc1], [na2, nb2, nc2]]
```

```
229
230     def mutarIndividuos(self, cromosoma):
231         #--- Segmeta el cromosoma para cada metabolito ---#
232         p1 = np.array(list(cromosoma[0]))
233         p2 = np.array(list(cromosoma[1]))
234         p3 = np.array(list(cromosoma[2]))
235         #--- Punto aleatorio del cromosoma para mutar ---#
236         pMutacion = np.random.randint(low=0, high=3)
237         #--- PROCESO DE MUTACIÓN ---#
238         #--- Cambia el valor del bit en el punto P ------#
239         #--- Para cada metabolito -----------------------#
240         if p1[pMutacion] == '0':
241             p1[pMutacion] = '1'
242         else:
243             p1[pMutacion] = '0'
244
245         pMutacion = np.random.randint(low=0, high=3)
246         if p2[pMutacion] == '0':
247             p2[pMutacion] = '1'
248         else:
249             p2[pMutacion] = '0'
250
251         pMutacion = np.random.randint(low=0, high=3)
252         if p3[pMutacion] == '0':
253             p3[pMutacion] = '1'
254         else:
255             p3[pMutacion] = '0'
256
257         pn1 = "".join(p1)
258         pn2 = "".join(p2)
259         pn3 = "".join(p3)
260         individuoMutado = []
261         individuoMutado.append(pn1)
262         individuoMutado.append(pn2)
263         individuoMutado.append(pn3)
264         #--- Regresa un nuevo individuo mutado ---#
265         return individuoMutado
266
267     def graficarBiomasa(self):
268         x = [i[0] for i in self.data]
269         y = self.arrayBiomass
270         plt.plot(x, y)
271         plt.title("Maximizar Biomasa respecto a las generaciones")
272         plt.xlabel("Generación")
273         plt.ylabel("Biomasa")
274         plt.show()
275
276     def graficarAptitud(self):
277         x = [i[0] for i in self.data]
278         y = self.arrayCond
279         plt.plot(x, y)
280         plt.title("Funcion Evaluación por Generación")
281         plt.xlabel("Generación")
282         plt.ylabel("Condicion ")
283         plt.grid()
284         plt.show()
285
286     def graficarConcentracionNutrientes(self):
287         x = [i[0] for i in self.data]
288         y1 = self.arrayCO2
289         y2 = self.arrayO2
290         y3 = self.arrayAC
291
292         plt.plot(x, y1, label = 'CO2')
293         plt.plot(x, y2, label = 'O2')
294         plt.plot(x, y3, label = 'AC')
295         plt.title("Minimizar  concentracion de reactivos")
296         plt.xlabel("Generación")
297         plt.ylabel("Concentración [mmol/gDw/H]")
298         plt.legend()
299         plt.grid()
300         plt.show()
```

```
301
302
303 #---- Sección de llamadas a las funciones ---#
304 if __name__ == "__main__":
305     tiempoInicial = time.time()                          #--- Registro del tiempo INICIAL de ejecución ----------#
306     algoritmoGenetico = AlgoritmoGenetico()              #--- Crea objeto de tipo AlgoritmoGenetico -------------#
307     algoritmoGenetico.generarPrimerPoblacion()           #--- Genera la primera población -----------------------#
308     algoritmoGenetico.reproduccion()                     #--- Inicia proceso de reproducción --------------------#
309     tiempoFinal = time.time() - tiempoInicial            #--- Registro del tiempo FINAL de ejecución ------------#
310     print("Tiempo De Prediccion:", tiempoFinal)          #--- Visualiza tiempo de ejecución de programa ----------#
311     algoritmoGenetico.graficarConcentracionNutrientes()  #--- Visualiza gráfica de concentracion de nutrientes ---#
312     algoritmoGenetico.graficarBiomasa()                  #--- Visualiza gráfica optimización de biomasa ----------#
313     algoritmoGenetico.graficarAptitud()                  #--- Visualiza gráfica optimización de aptitud ----------#
```

Código de programa que tiene como función utilizar la librería COBRApy

```
1  #---Importar Librerías ----#
2  import cobra.test
3
4  #--- Modelo metabólico ---#
5  model = cobra.test.create_test_model("textbook")
6
7  #--- Metabolitos ---#
8  METABOLITE_1 = "EX_co2_e"
9  METABOLITE_2 = "EX_o2_e"
10 METABOLITE_3 = "EX_ac_e"
11
12 #--- Función optimización ---#
13 def evaluarModeloCOBRApy(x, y, z):
14     #--- Límites de concentración ---#
15     model.reactions.get_by_id(METABOLITE_1).upper_bound = x
16     model.reactions.get_by_id(METABOLITE_2).upper_bound = y
17     model.reactions.get_by_id(METABOLITE_3).upper_bound = z
18     #--- Regresa el valor de Biomasa a partir de ---#
19     #--- las concentraciones de los nutrientes -----#
20     return model.optimize().objective value
```

BIBLIOGRAFÍA

[1] A. W. Services, Amazon Machine Learning: Guía del desarrollador, 2019.

[2] V. Mathivet, Inteligencia artificial para desarrolladores; conceptos e implementaciones en java, Ediciones ENI, 2017.

[3] A. G. Josh Patterson, Deep Learning A Practitioner´s Approach, O'Reilly, 2017.

[4] L. G. F, Artificial intelligence : structures and strategies for complex problem solving, ADDISON WESLEY PUB CO INC, 2008.

[5] T. M. Michell, Machine Learning, McGraw-Hill, 1997.

[6] P. Ponce, Inteligencia artificial con aplicaciones a la ingeniería, México: Alfaomega Grupo Editor.

[7] H. J. L. S. A. Hartwell LH, «From molecular tomodular cell biology,» *Nature,* vol. 402, nº C47–C52, 1999.

[8] V. Donalt, Fundamentos de bioquímica; la vida a nivel molecular, Buenos aires: Médica Panamericana, 2009.

[9] J. D. A. N. J. Kinross, «Gut microbiome-host interactions in health and disease,» *Genome Med,* vol. 3, nº 14, 2011.

[10] H. U. K. T. Y. &. L. S. Y. Kim, «Metabolic flux analysis and metabolic engineering of microorganisms,» *Mol. BioSyst,* vol. 4, nº 2, pp. 113-120, 2008.

[11] M. W. X. N. C. T. J. &. K. J. R. Covert, «Integrating metabolic, transcriptional regulatory and signal transduction models in Escherichia coli.,» *Bioinformatics,* vol. 24, nº 18, pp. 2044-2050, 2008.

[12] B. a. A. a. J. a. A. M. P. Merino, «Stoichiometric model and metabolic flux analysis for Leptospirillum ferrooxidans,» *Biotechnol. Bioeng,* vol. 107, nº 4, pp. 696-706, 2010.

[13] D. C. Romero Juan Jesús, Inteligencia Artificial y computación avanzada, Colección Informática, 2007.

[14] F. T. Jaime, Bioquímica: La ciencia de la vida, San José C.R: EUNED, 2007.

[15] M. G.-G. P. P. d. P. G. M. L. E. S. S. &. R.-P. A. Gutiérrez, «A New Improved and Extended Version of the Multicell Bacterial Simulator gro,» *ACS Synthetic Biology,* vol. 6, nº 8, p. 1496–1508, 2017.

[16] A. L. J. A. P. B. O. &. H. D. R. Ebrahim, «COBRApy: Constraints-Based Reconstruction and Analysis for Python,» *BMC Systems Biology,* vol. 7, nº 1, p. 74, 2013.

[17] H. C. R. J. K. M. J. A. K. P. B. L. H. V. P. B. Feist AM, «A genome-scale metabolic reconstruction for Escherichia coli K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information,» *Mol SystBiol,* vol. 3, p. 121, 2007.

[18] J. D. C. T. M. N. J. L. J. A. N. H. F. A. M. &. P. B. O. Orth, «A comprehensive genome-scale reconstruction of Escherichia coli metabolism,» *Molecular Systems Biology,* vol. 7, nº 1, p. 535–535, 2014.

[19] F. D.-L. G. C. G. G. Rosiñol Ricardo, «El uso de herramientas de simulación basadas en agentes en sistemas biológicos,» 2013.

[20] G. P. M. Eduardo, «A new agent-based platform for simulating,» Madrid, España.

[21] L. D. M. H. T. Y. Oyetunde T, «Machine learning framework for assessment of microbial factory performance,» *PloS one,* nº 14, 2019.

[22] W. J. S. P. Savage DF, «Defossiling fuel: how synthetic biology can,» *ACS Chem Biol,* vol. 3, pp. 13-16, 2008.

[23] L. M.-N. M. &. R. D. A. Dethlefsen, «An ecological and evolutionary perspective on human–microbe mutualism and disease,» *Nature,* vol. 449, pp. 811-818, 2007.