

UNIVERSIDAD DE GUANAJUATO
CAMPUS GUANAJUATO
DIVISIÓN DE CIENCIAS NATURALES Y EXACTAS



“MODELADO Y SIMULACIÓN DINÁMICA DE UN SISTEMA
DE DESTILACIÓN INTENSIFICADO PARA LA PREDICCIÓN
DE LA COMPOSICIÓN DE LOS PRODUCTOS MEDIANTE
REDES NEURONALES”

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO QUÍMICO

PRESENTA:

ABRAHAM RODARTE DE LA FUENTE

GUANAJUATO, GTO., A 30 DE MARZO DE 2023

DECLARATORIA

Por medio de la presente me responsabilizo de la autenticidad y originalidad del presente trabajo titulado:

“MODELADO Y SIMULACIÓN DINÁMICA DE UN SISTEMA DE DESTILACIÓN INTENSIFICADO PARA LA PREDICCIÓN DE LA COMPOSICIÓN DE LOS PRODUCTOS MEDIANTE REDES NEURONALES”

Dr. Juan Gabriel Segovia Hernández
Director de Tesis

Dr. Eduardo Sánchez Ramírez
Co- Director de Tesis

Dr. Esteban Abelardo Hernández Vargas
Co- Director de Tesis

GUANAJUATO, GTO., A 30 DE MARZO DE 2023

Miembros del Jurado del Examen Profesional que para obtener el título de Ingeniero Químico presenta el C. Abraham Rodarte de la Fuente, con el trabajo titulado:

“MODELADO Y SIMULACIÓN DINÁMICA DE UN SISTEMA DE DESTILACIÓN INTENSIFICADO PARA LA PREDICCIÓN DE LA COMPOSICIÓN DE LOS PRODUCTOS MEDIANTE REDES NEURONALES”

Dr. Salvador Hernández Castro
Presidente

Dr. Fernando López Caamal
Secretario

Dr. Guillermo Barrios del Valle
Vocal

GUANAJUATO, GTO., A 30 DE MARZO DE 2023

GUANAJUATO, GTO., A 30 DE MARZO DE 2023

“MODELADO Y SIMULACIÓN DINÁMICA DE UN SISTEMA DE DESTILACIÓN INTENSIFICADO PARA LA PREDICCIÓN DE LA COMPOSICIÓN DE LOS PRODUCTOS MEDIANTE REDES NEURONALES”

Bajo la dirección de: Dr. Juan Gabriel Segovia Hernández

Presenta: Abraham Rodarte de la Fuente

RESUMEN

En este trabajo se presenta el modelado y la simulación dinámica, mediante redes neuronales, del sistema de destilación intensificado (columna de pared divisoria) para separar la mezcla acetona-etanol-butanol obtenida en un proceso de fermentación. Obteniendo resultados similares al comparar la capacidad de predicción de los datos obtenidos en el simulador ASPEN Dynamics a partir de una optimización estructural de las redes neuronales desarrolladas en el lenguaje de programación Python 3.8.

La dinámica de la columna de pared divisoria de este caso de estudio puede ser modelada por una red neuronal y el análisis de la estructura de la red neuronal tiene un gran impacto en la capacidad de predicción, determinando una red tipo LSTM (neurona con memoria a corto plazo) con la función de activación lineal, el optimizador RMSPROP (propagación de la media cuadrática), una sola capa oculta y cinco neuronas en ella, alimentando solo dos variables manipulables de las tres con las que cuenta el sistema pudiendo predecir el perfil de composición de los compuestos purificados.

ABSTRACT

This work presents the modeling and dynamic simulation, by means of neural networks, of the intensified distillation system (dividing wall column) to separate the acetone-ethanol-butanol mixture obtained in a fermentation process. Obtaining good results when comparing the prediction capacity of the data obtained in the ASPEN Dynamics simulator from a structural optimization of the neural networks developed in the Python 3.8 programming language.

The dividing wall column dynamics of this case study can be modeled by a neural network and the analysis of the neural network structure has a great impact on the prediction capability, determining a LSTM (Long short-term memory) type network with the linear activation function, the RMSPROP (root mean square propagation) optimizer, a single hidden layer and five neurons in it, feeding only two manipulable variables out of the three available in the system and being able to predict the composition profile of the purified compounds.

DEDICATORIA

Dedico el presente trabajo a mis padres Samuel Rodarte García y Olimpia del Carmen de la Fuente Martínez por todo el sacrificio que realizaron a lo largo de los años para que yo pudiera salir adelante.

AGRADECIMIENTOS

A mis padres por apoyarme en todo momento para poder salir adelante y cumplir mis sueños.

A mis hermanas Samantha y Laura, por todo el amor que me han dado durante toda mi vida.

A todos los amigos que tuve a lo largo de la carrera que me hicieron tan feliz y me ayudaron a crecer como persona.

Debo agradecer la paciencia y el apoyo del Dr. Juan Gabriel Segovia Hernández por creer en mi trabajo durante todo el proceso.

NOMENCLATURA

ABE	Producción biotecnológica de acetona, butanol y etanol
ADAM	Estimación adaptiva del momento
AIC	Valor del criterio de información de Akaike
Btu	Unidad térmica británica
b_k	Polarización de la neurona k
CNN	Red neuronal convolucional
CPU	Unidad de procesamiento central
DETL	Evolución diferencial con lista tabú
DWC	Columna de pared divisoria
GPU	Unidad de Procesamiento Gráfico
hr	Horas
lb	Libras
Lineal	Función de activación lineal
$\log ()$	Logaritmo base 10
LSTM	Memoria a corto plazo
MCP	Modelo de Control Predictivo
MSE	Error Cuadrático Promedio
n	Número de datos
NRTL-HOC	Modelo no aleatorio de dos líquidos Hayden-O'Connell
PID	Controlador proporcional, integral y derivativo
ReLU	Función de activación unidad lineal rectificadora
RMSPROP	Propagación de la raíz cuadrada
RNA	Red neuronal artificial
s_k	Señal de salida de la neurona k
SGD	Descenso de gradiente estocástico
SSE	Suma de cuadrados de los errores
Tanh	Función lineal tangente hiperbólica
u_k	Combinación lineal de las entradas ponderadas
w_{kj}	Conjunto de pesos sinápticos de la neurona k
x_j	Conjunto de señales de entrada
y_i	Valor obtenido de la simulación
\bar{y}_i	Valor predicho por la RNA
y_{scaled}	Valor normalizado
Σ	Sumatoria
φ	Función de activación

ÍNDICE

RESUMEN	4
ABSTRACT	5
DEDICATORIA	6
AGRADECIMIENTOS	7
NOMENCLATURA	8

CAPÍTULO 1 "INTRODUCCIÓN"

1.1 INTRODUCCIÓN	13
Tabla 1.1. Ventajas y desventajas de las RNA.	14
1.2 BREVE INTRODUCCIÓN BIOLÓGICA	14
Figura 1.1. Componentes de una neurona biológica.	15
1.3 HISTORIA	15
1.4 APLICACIONES Y AVANCES DE LAS REDES NEURONALES EN LA INGENIERÍA QUÍMICA	16
1.5 JUSTIFICACIÓN	17
1.6 HIPÓTESIS	18
1.7 OBJETIVO GENERAL	18

CAPÍTULO 2 "DESTILACIÓN"

2.1 DESTILACIÓN	20
Figura 2.1. Columna de destilación convencional.	20
Figura 2.2. Gráfica de costo total mínimo para la separación de una mezcla benceno-tolueno	21
2.2 DESTILACIÓN INTENSIFICADA	22
2.3 COLUMNA DE PARED DIVISORIA	22
Figura 2.3. Columna de pared divisoria del proceso de obtención de biocombustibles de (Errico et al., 2017).	23
2.4 MODELADO Y CONTROL	24
Figura 2.4. Modelado de una columna de destilación convencional con redes neuronales de (Shin et al., 2020b).	25
Figura 2.5. Resultados obtenidos de (Shin et al., 2020b).	25
Figura 2.6. Sistema propuesto en (de Araújo Neto et al., 2021).	26

CAOPITULO 3 "REDES NEURONALES"

3.1 REDES NEURONALES	28
Figura 3.1. Estructura jerárquica de una RNA.	28
Figura 3.2. Capas de una RNA.	28
Figura 3.3. Funciones de activación.	30
3.2 ARQUITECTURA DE LAS RNA	31
3.3 REDES FEED-FORWARD	31
3.4 REDES RECURRENTE	32
3.5 RED NEURONAL TIPO LSTM.	32

<i>Figura 3.4. Estructura de una red tipo LSTM.</i>	33
<i>Figura 3.5. Estructura secuencial de una neurona tipo LSTM.</i>	33

CAPÍTULO 4 "METODOLOGÍA"

4.1 METODOLOGÍA	36
<i>Figura 4.1. Estructura para el caso del sistema intensificado.</i>	37
<i>Figura 4.2. Ejemplo de división de datos obtenidos del perfil de composición del butanol para prueba y entrenamiento en la columna intensificada.</i>	37
<i>Figura 4.3. Determinación del número de épocas utilizadas.</i>	38
4.2 OPTIMIZACIÓN ESTRUCTURAL.....	38
<i>Figura 4.4. Cambios en la estructura de la RNA para poder obtener el mejor modelo en el caso intensificado.</i>	38
4.3 NÚMERO DE CAPAS OCULTAS.....	39
4.4 PREDICCIÓN	39
4.5 CASO DE ESTUDIO	39
<i>Figura 4.5. Caso de estudio.</i>	40
Tabla 4.1. Flujo de alimentación.	40
Tabla 4.2. Parámetros de diseño para la secuencia considerada como caso de estudio.	41
<i>Figura 4.6. Generación de datos del sistema intensificado: a) Variables Manipulables y b) Variables Perturbadas.</i>	42
<i>Figura 4.7. Matriz de cofactores de los datos obtenidos de la columna de pared divisoria.</i>	42

CAPÍTULO 5 "RESULTADOS"

5.1 RESULTADOS.....	45
<i>Figura 5.1. Resultados de la capacidad de predicción con diferentes funciones de activación usando el optimizador RMSPROP para el sistema intensificado: a) Comparación, b) Función Tanh, c) Lineal y d) ReLU.</i>	45
<i>Figura 5.2. Valor obtenido de AIC con diferentes funciones de activación usando el optimizador RMSPROP para el sistema intensificado: a) Comparación, b) Función Tanh, c) Lineal y d) ReLU.</i>	46
<i>Figura 5.3. Resultados de la predicción realizada para el sistema intensificado usando la función Lineal, el optimizador RMSPROP y 5 neuronas en la capa oculta: a) Acetona, b) Etanol y c) Butanol.</i>	47
5.2 ANÁLISIS DE RESULTADOS DE LA RNA CON UNA SOLA CAPA OCULTA.....	48
Tabla 5.1 Comparación de optimizadores en las estructuras con el valor de AIC más bajo obtenido para la columna intensificada.	48
5.3 IDENTIFICACIÓN DE VARIABLES CON MÁS IMPORTANCIA-PESO EN EL MODELO DE LA RNA	48
<i>Figura 5.4 Resultados de la capacidad de predicción con diferentes funciones de activación usando el optimizador de RMSPROP para el sistema intensificado.</i>	49
<i>Figura 5.5 Valor obtenido de AIC con diferentes funciones de activación usando el optimizador RMSPROP para el sistema intensificado: a) Comparación, b) Función Tanh, c) Lineal y d) ReLU.</i>	49
<i>Figura 5.6 Predicción realizada alimentando la relación de reflujo y la carga térmica.</i>	50
<i>Figura 5.7 Diagrama de las variables con más peso alimentadas a la estructura de la RNA.</i>	50
5.4 DETERMINACIÓN DEL NÚMERO DE CAPAS OCULTAS	51
<i>Figura 5.8 Comparación del valor del MSE al aumentar el número de capas ocultas a la RNA.</i>	51
<i>Figura 5.9 Comparación del valor del AIC al aumentar el número de capas ocultas.</i>	51
5.5 ANÁLISIS DE RESULTADOS PARA EL SISTEMA INTENSIFICADO	52

<i>Figura 5.10 Evolución del valor del MSE al aumentar el número de capas ocultas con 5 neuronas en la capa final.</i>	52
Tabla 5.2. Resumen de la estructura de la RNA con mejor capacidad de modelado para el sistema intensificado.	52

CAPÍTULO 6 "CONCLUSIONES"

6.1 CONCLUSIONES	54
------------------------	----

CAPÍTULO 7 "RECOMENDACIONES"

7.1 RECOMENDACIONES	56
---------------------------	----

CAPÍTULO 8 "APÉNDICE A"

8.1 APÉNDICE A	58
----------------------	----

CAPÍTULO 9 "REFERENCIAS"

9.1 REFERENCIAS	68
-----------------------	----

CAPÍTULO 1

INTRODUCCIÓN

1.1 INTRODUCCIÓN

En los últimos años se ha innovado en el uso de la inteligencia artificial para resolver una gran cantidad de problemas complejos (Behrad and Saniee Abadeh, 2022). Por su parte el aprendizaje profundo se puede identificar como un proceso que se utiliza para la resolución de estos problemas; entre ellos, el reconocimiento de voz, la clasificación de imágenes y la transformación de texto a voz (Manickam et al., 2021). Por su parte el *machine learning* o aprendizaje automático se ha convertido en una tecnología apasionante de los últimos tiempos llevando a los científicos a innovar en la implementación de estos recursos para reducir costos del control en una columna de destilación (Shin et al., 2020a) y hasta prevenir enfermedades como la detección del cáncer a partir de los síntomas que presenten los pacientes (Tsai et al., 2017).

El aprendizaje automático es aplicado a una inteligencia artificial para que pueda procesar grandes volúmenes de información y generar así análisis y resultados (Deng et al., 2022). El objetivo de construir máquinas con inteligencia artificial es poder realizar las funciones de diagnóstico, predicción y reconocimiento, aprendiendo del conjunto de datos de entrenamiento que se le proporciona. Se pueden usar datos de muestra para entrenar a la máquina y predecir el resultado deseado (Hüsken and Stagge, 2003).

También una de las ventajas del *machine learning* es la identificación de patrones dentro de los datos de entrenamiento produciendo resultados para cualquier entrada similar que se le proporcione a la inteligencia artificial (Yildirim et al., 2011). Debido a la fuerte conexión entre el aprendizaje automático y la ciencia de datos, procesamiento de grandes volúmenes de datos y relacionamiento entre las variables subyacentes, es posible obtener información importante sobre grandes volúmenes de datos (Wang, 2022). Otra ventaja importante es el mejoramiento de la eficiencia, ya que una máquina con este aprendizaje aprenderá a adaptarse y mejorará su propia eficiencia independientemente de su entorno. Una inteligencia artificial puede completar una tarea específica sin olvidar ningún paso que se incluyen en el programa. Finalmente, trae consigo la capacidad de manejar múltiples dimensiones y variedades de datos simultáneamente y en condiciones inciertas como cuando se utiliza las variables medibles del clima para poder predecir la potencia del viento usando redes neuronales (Ookura and Mori, 2020).

Es difícil interpretar los resultados, de sistema modelado con redes neuronales, con precisión para determinar la efectividad del algoritmo de aprendizaje automático. Para ello se debe experimentar con diferentes algoritmos antes de elegir uno para entrenar la máquina (Deng et al., 2022).

También es importante el lenguaje de programación utilizado para el desarrollo del código ya que debe adaptarse al clima tecnológico cambiante y el desarrollo de nuevos lenguajes de programación para comunicar estos avances tecnológicos, es casi imposible convertir los programas y sistemas en nuevas sintaxis (Cardozo and Mens, 2022). Una alternativa es el lenguaje de programación Python para implementar los algoritmos de manera versátil, usado para resolver problemas de ciencias e ingenierías (Vadakara and Basu, 2019). Python fue creado por Guido Van Rossum y es de código abierto, por lo que cualquiera puede modificarlo y contribuir en su desarrollo, siendo el más usado para desarrollo de inteligencia artificial contando con una gran cantidad de librerías.

Las redes neuronales artificiales (RNA) hacen que las inteligencias artificiales puedan procesar información de la misma manera que el cerebro humano, una característica muy importante de estas redes es cómo se estructuran las capas para procesar la información (Keenan et al., 2022). Cada capa consiste en una red de neuronas interconectadas que procesan la información alimentada para poder resolver un problema. Una máquina con arquitectura de red neuronal aprende con el ejemplo realizando ajustes en cada una de las neuronas que constituyen la red.

Los sistemas inteligentes deben ser eficientes en la toma de decisiones en un entorno incierto debido a su capacidad de realizar diversas operaciones (KLIR, 1997). Las redes neuronales RNA surgieron como una rama central de la inteligencia artificial y desde hace más de dos décadas se han aplicado con éxito a sistemas de control, esencialmente en la modelación, predicción de sistemas no lineales y en el control de procesos químicos (Azlan Hussain, 1999).

Gracias a los avances en el control basado en modelos, se debe tener conocimiento sobre las ventajas y desventajas que aportan las RNA en la modelación las cuales se enlistan en la Tabla 1.1.

Tabla 1.1. Ventajas y desventajas de las RNA (Hochreiter and Schmidhuber, 1997).

Ventajas	Aprendizaje adaptativo en el cual se aprende a realizar tareas a partir de un conjunto de datos representados como pesos y entradas.
	Operación en tiempo real, que pueden ser llevadas a cabo por computadoras o dispositivos de hardware especial para aprovechar su capacidad.
Desventajas	Elevado tiempo de aprendizaje que depende del número de patrones a reconocer.
	No tiene la capacidad para interpretar los resultados que esta produce siendo necesario la interpretación del desarrollador.

1.2 BREVE INTRODUCCIÓN BIOLÓGICA

Varias estimaciones indican que el sistema nervioso contiene alrededor de cien mil millones de neuronas que al ser observadas con el microscopio tienen múltiples formas, aunque la mayoría presenta un cuerpo celular de entre 10 y 80 micras de longitud, de donde surge un denso árbol de ramificaciones compuesto por dendritas y del cual se extiende una fibra tubular llamada axón que también se ramifica en su extremo final para conectar con otras neuronas, Fig. 1.1, (Vepa, 2009).

Se podría decir que las neuronas constituyen procesadores de información sencillos constituidos por un canal de entrada de información, dendritas, un órgano de cómputo, soma y el axón, que sería el canal de salida, donde se envía la señal de salida a otras neuronas. La

sinapsis es la unión entre dos neuronas que permanecen separadas por un pequeño vacío de unas 0.2 micras. Siendo las neuronas presinápticas las que envían las señales y postsinápticas las que las reciben. La sinapsis es unidireccional, por lo que la información fluye siempre en un único sentido (Conte et al., 2006).

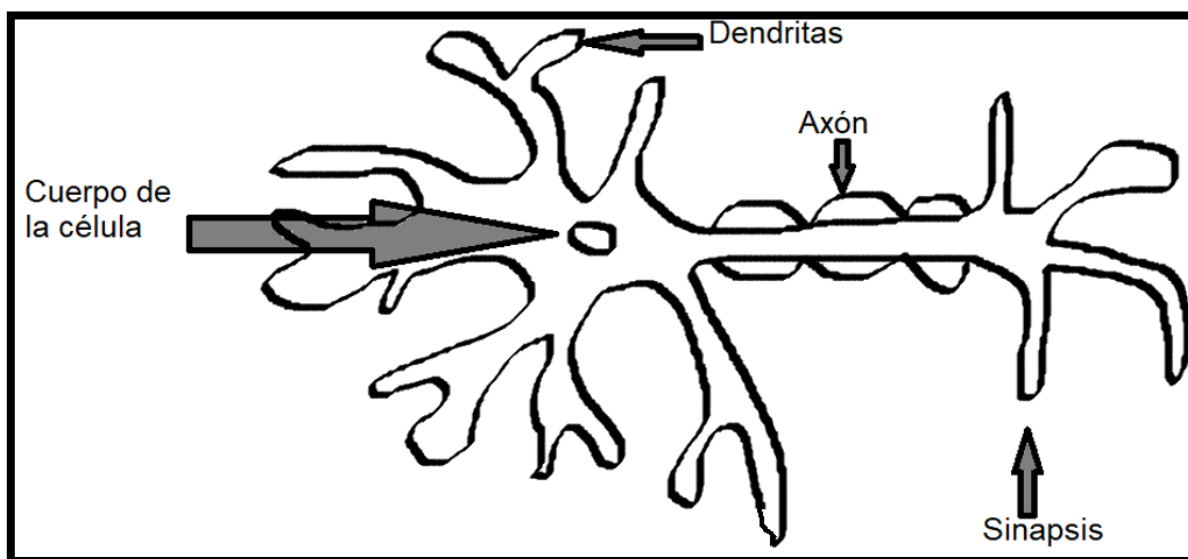


Figura 1.1. Componentes de una neurona biológica.

La intensidad de una sinapsis puede ser modulada en una escala temporal mucho más amplia que la del disparo de las neuronas, esta plasticidad sináptica constituye en buena medida el aprendizaje. Durante el desarrollo de los seres vivos el cerebro se moldea por lo que muchas cualidades del individuo se adquieren por la influencia de la información que del medio externo proporciona sus sensores por diferentes formas: por el establecimiento de nuevas conexiones, rupturas de otras, modelado de las intensidades sinápticas o incluso a través de la muerte neuronal (Zetterberg, 2018).

1.3 HISTORIA

- 1936: Alan Turing estudia por primera vez el cerebro como una forma de ver el mundo de la computación (Graham-Cumming, 2012). Sin embargo, los primeros en dar una fundamentación a la computación neuronal fueron McCulloch y Walter Pitts (Wilson, 2008).
- 1943: McCulloch y Walter Pitts lanzan una teoría acerca de la forma de trabajar de las neuronas (Wilson, 2008).
- 1949: Donald Hebb explica por primera vez sobre el problema del aprendizaje elaborando una norma de como el aprendizaje ocurría y que esto se daría cuando ciertos cambios en una neurona eran activados (Cooper, 2005).
- 1950: Karl Lashley comenta que la información es distribuida en el cerebro y no es centralizada (Lanska, 2014).
- 1956: Minsky, McCarthy, Rochester y Shanon desarrollaron una conferencia de inteligencia artificial donde se hace la primera toma de contacto con las redes neuronales artificiales (Hewett and Marcus, 2002).

- En 1957 Frank Rosenblatt inventa el perceptrón en el laboratorio de Cornell, basándose en los primeros conceptos de neuronas artificiales (Wilson, 2008).
- 1959-1960: Bernard Widrow crea un sistema lineal y adaptativo que denomina Adaline y en la que implementa dos capas llamadas Madaline, que se utiliza para conseguir un reconocimiento de voz, caracteres o para eliminar el eco que se producía en las líneas telefónicas (Rogers, 1997).
- 1969: Marvin Minsky y Seymour Papert prueban matemáticamente que el perceptrón no era capaz de resolver problemas como el aprendizaje de una función no lineal. Problema de gran importancia ya que las funciones no lineales son extensamente empleadas, lo que causó que se tuviera una pérdida de confianza en este nuevo campo (Hewett and Marcus, 2002).
- En la actualidad numerosos trabajos de redes neuronales artificiales se realizan y publican cada año modelando complejos sistemas no lineales. Su resurgimiento se debe a la dificultad para resolver con la eficiencia deseada problemas como los de visión o aprendizaje (D. Liu et al., 2022).

1.4 APLICACIONES Y AVANCES DE LAS REDES NEURONALES EN LA INGENIERÍA QUÍMICA

Los sistemas inteligentes deben tomar decisiones eficaces y racionales sobre las capacidades de aprendizaje automático, procesamiento de información en paralelo, generalización y otras operaciones en entornos inciertos. Dado su enorme potencial, las redes neuronales artificiales se han convertido en una rama central de la inteligencia artificial. Especialmente debido al rápido desarrollo de computadoras de alta velocidad e integración a gran escala, y la creciente investigación sobre algoritmos y arquitecturas de sistemas neuronales. Una de estas aplicaciones atractivas son los sistemas de control inteligente, que se utilizan principalmente para el modelado y predicción de sistemas dinámicos no lineales e incompletos, y la adaptación y control de estos sistemas.

Los retos que se presentan son el llevar a cabo el control y el modelado de sistemas de separación como es el caso de la destilación, en todas sus variantes. Las oportunidades que presentan estas herramientas son de gran importancia para la reducción de costos de la operabilidad de equipos complejos, lo cual es una de las metas que se tiene al realizar la optimización del control del proceso. Aplicando estos controladores a equipos que buscan reducir la cantidad de energía en la separación de compuestos de gran demanda como los combustibles, crean un sistema rentable y amigable con el medio ambiente.

Las redes neuronales son excelentes candidatas para resolver una amplia gama de problemas en la industria química. Estos incluyen clasificación de materias primas multicomponentes, reconocimiento de patrones y análisis de composición química (Kakkar et al., 2021), interpretación cualitativa de datos de proceso, control adaptativo, detección de fallas de sensores, modelado, caracterización y optimización de unidades de proceso, modelado de fenómenos entendidos como flujo turbulento.

Las redes neuronales analizan los datos del sistema y generan modelos internos relacionados con los datos a través de un proceso de aprendizaje. Este modelo interno no se basa en ninguna especificación de los mecanismos subyacentes del proceso. Para la generación de modelos, no se requiere conocimiento previo de los principios subyacentes del proceso o fenómeno que se está modelando. Esta propiedad los hace ideales para modelar sistemas

complejos donde convergen la transferencia de calor, la mecánica de fluidos, la transferencia de masa, la cinética y los fenómenos catalíticos, y aunque pueden describirse mediante ecuaciones diferenciales, pueden presentar problemas de solución debido a su no linealidad (Moon et al., 2022). También permiten invertir en la operación de un modelo simulado, mientras que los modelos tradicionales determinan el valor de una condición de salida o variable dependiente a partir de un conjunto de datos de entrada o variables independientes, las redes neuronales permiten el cálculo de condiciones de entrada dado un conjunto de salidas de los datos del sistema (Srinivasan, 2022).

Los algoritmos genéticos pueden asimilarse al proceso biológico de la evolución natural de ciertos rasgos de una especie que se transmiten de generación en generación y contribuyen a su supervivencia y mejora de la población a medida que avanza el ciclo evolutivo; los algoritmos genéticos funcionan de la misma manera. El algoritmo comienza con una muestra aleatoria de soluciones y trabaja a través de su proceso para generar una nueva generación de soluciones hasta que se cumplan los criterios de optimización (Poznyak et al., 2019).

En los últimos años, diferentes autores (de Araújo Neto et al., 2021; El-Gendy et al., 2019; Neves et al., 2021; Shin et al., 2020b) han realizado estudios comparando el uso de controladores basados en inteligencia artificial con los usados típicamente en la industria, obteniendo buenos resultados al poder realizar una predicción más precisa mediante el Modelo de Control Predictivo (MCP) introduciéndose a la industria para poder aumentar la eficiencia operacional (Shin et al., 2020b). Además, con el manejo de software de simulación se permite interactuar con la inteligencia artificial para auxiliarse en el diseño, optimización y administración de las plantas químicas (Sánchez-Ramírez et al., 2020).

Con esto en mente, en el presente trabajo se propone una metodología para modelar sistemas intensificados. Este modelo se realiza mediante el uso de redes neuronales por la naturaleza de la mezcla, siendo termodinámicamente muy compleja y con presencia de azeótropos, que le da un problema de mucha no linealidad a la dinámica del sistema; tomando también en cuenta que se modela con modelos termodinámicos complejos, por lo que una red neuronal nos puede facilitar esto.

1.5 JUSTIFICACIÓN

Debido a la complejidad que representa el modelado de la dinámica de procesos intensificados, se necesitan herramientas novedosas que tengan la misma capacidad de predicción que modelos complejos, pero con ventajas computacionales que permite una aplicación más directa, por ejemplo, las redes neuronales.

La automatización y el control de procesos actualmente toman un papel muy importante en las mejoras de la producción, teniendo una evolución en los algoritmos de programación que se están utilizando. Las RNA están siendo aplicadas en el campo de identificación de sistemas, predicción de procesos y la teoría de control, por su gran capacidad de optimización que tiene en procesos no lineales o en la facilidad que tienen para encontrar los parámetros óptimos de control.

Estas tecnologías se presentan como herramientas que aprovechan el gran poder de cómputo con el que se cuenta actualmente para la resolución de problemas de ingeniería que, con sistemas de ecuaciones complejos, como al determinar el modelo de la dinámica de un proceso industrial.

1.6 HIPÓTESIS

El cambio de las variables medibles de los sistemas de destilación intensificados puede ser modelada y simulada utilizando redes neuronales artificiales para poder predecir su comportamiento.

1.7 OBJETIVO GENERAL

Diseñar una red neuronal para predecir la dinámica de una columna de destilación intensificada (columna de pared divisoria) a partir de los datos obtenidos en un simulador.

CAPÍTULO 2

DESTILACIÓN

2.1 DESTILACIÓN

En la destilación una mezcla de alimentación de dos o más componentes se separa en dos o más productos, incluidos, entre otros, los productos del domo y del fondo, cuya composición es diferente de la composición de la alimentación (Gómez-Castro et al., 2008). La mayoría de las veces, la alimentación es un líquido o una mezcla de vapor y líquido. Los fondos casi siempre son líquidos, pero el destilado puede ser líquido, vapor o ambos. Los requisitos de separación son que se forme una segunda fase para que tanto el líquido como el vapor estén presentes y puedan estar en contacto mientras fluyen a contracorriente entre sí en platos o columnas empacadas, de modo que los componentes tengan diferente volatilidad para que se puedan separar diferentes grados de fases, y las dos fases se pueden separar por gravedad o por medios mecánicos. La destilación difiere de la absorción y la extracción en que la segunda fase fluida generalmente se crea térmicamente (evaporación y condensación) en lugar de introducir una segunda fase que puede contener componentes adicionales que no están presentes en la mezcla, Fig 2.1.

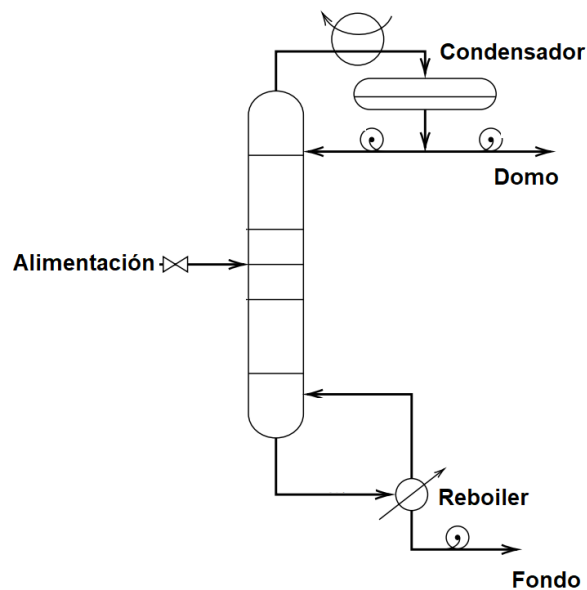


Figura 2.1. Columna de destilación convencional.

Los factores que influyen en el diseño o análisis de una columna de destilación son:

- Flujo de alimentación, composición, temperatura, presión y fase.
- Grado deseado de separación de los componentes.
- Presión de operación.
- Caída de presión.
- Relación de reflujo mínima y relación de reflujo real
- Número mínimo de etapas de equilibrio y número real de etapas de equilibrio (eficiencia de las etapas).
- Tipo de condensador (total, parcial o mixto).
- Grados de subenfriamiento del reflujo del líquido.
- Tipo de rehervidor (parcial o total).

- Tipo de platos o empaquetado.
- Altura de la columna.
- Etapa de alimentación.
- Diámetro de la columna.
- Materiales de construcción.
- Capacidad calorífica y reactividad química de los componentes de la alimentación.

La temperatura y la fase de la alimentación se determinan a la presión de la bandeja de alimentación mediante un cálculo de flash adiabático a través de la válvula de alimentación. A medida que aumenta la fracción de vapor de la alimentación, aumenta la relación de reflujo necesaria (L/D), pero disminuye la relación de ebullición (V/B).

A medida que se incrementa la presión de operación de la columna, las temperaturas en la columna aumentan, de manera similar a un gráfico de presión de vapor. La presión de funcionamiento de la columna en el condensador debe corresponder a una temperatura del destilado algo mayor (por ejemplo, de 6° a 28° C) que la temperatura de suministro del agua de refrigeración al condensador superior. Sin embargo, si esta presión se aproxima a la presión crítica del componente más volátil, deberá utilizarse una presión más baja y se requiere un refrigerante.

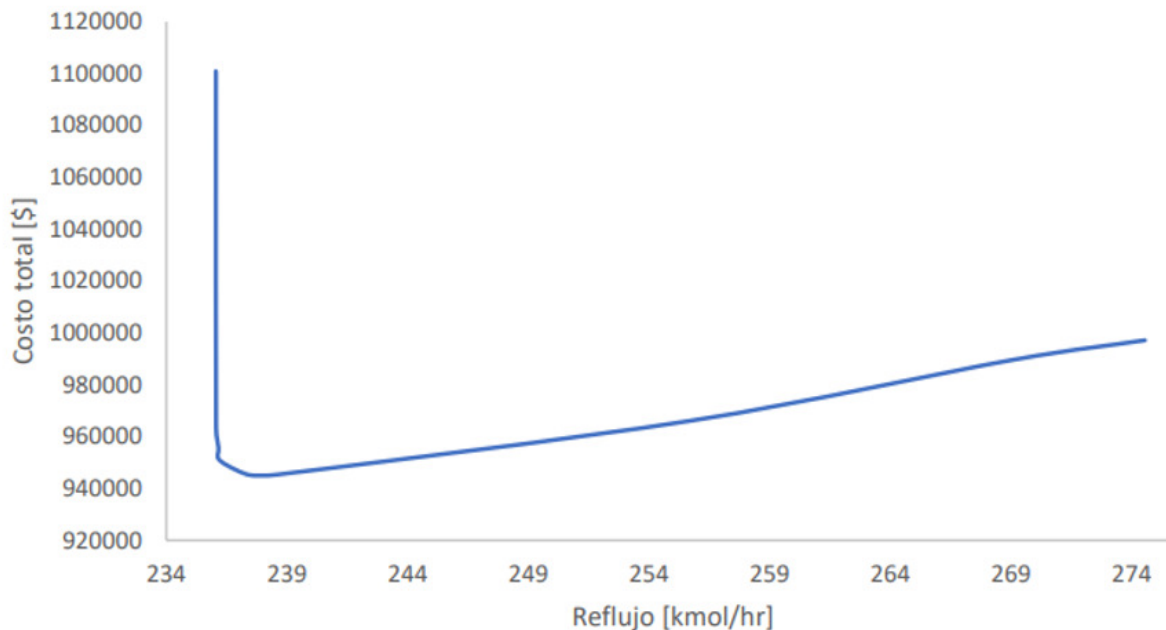


Figura 2.2. Gráfica de costo total mínimo para la separación de una mezcla benceno-tolueno.

2.2 DESTILACIÓN INTENSIFICADA

La intensificación de proceso es el desarrollo de equipos y técnicas innovadoras que ofrece mejoras en el proceso como la disminución del volumen del equipo, la reducción del número de equipos, consumo de energía, generación de residuos, dando lugar a tecnologías más baratas, seguras y sostenibles (Ponce-Ortega et al., 2012). La intensificación de procesos puede brindar significantes reducciones no solo en los equipos sino en la cantidad de energía que se consume. Se enfoca en el diseño innovador de procesos y equipos como sistemas multifuncionales, separaciones híbridas, así como la integración de energías alternativas. El diseño de nuevos procesos en la ingeniería química toma varios objetivos, incluyendo la minimización de los costos totales anuales y la maximización de las ganancias y productos generados.

La destilación multietapa es el método industrial más utilizado para separar mezclas químicas. Sin embargo, es una técnica que consume mucha energía, especialmente cuando la volatilidad relativa de los componentes clave que se separan es baja ($<1,50$). Se han propuesto sistemas de destilación acoplados térmicamente porque pueden realizar la tarea de separar mezclas con un bajo consumo de energía en comparación con los esquemas de destilación tradicionales. La estructura de estos sistemas complejos presenta desafíos de control debido a la transferencia de corrientes de vapor (o líquido) entre columnas. En particular, la presencia de bucles en estos sistemas acoplados crea la noción de que se pueden detectar algunos problemas de control durante la operación de estos sistemas en comparación con el buen desempeño de las secuencias de destilación convencionales.

2.3 COLUMNA DE PARED DIVISORIA

Estas columnas son completamente acopladas térmicamente en una coraza de una columna de destilación, donde es posible separar una corriente de alimentación de tres o más componentes. La columna es dividida por una pared en dos partes: el prefraccionador, que es donde entra la alimentación, y la columna principal, de donde sale la corriente lateral, como se observa en Fig 2.3.

Este arreglo tiene varias ventajas (Tian et al., 2022):

- Los arreglos con prefraccionador usualmente requieren de 20 a 30% menos energía que el arreglo convencional.
- Requiere menos costo capital que un arreglo de columnas simples.
- El material solo se recalienta una vez y su tiempo de residencia en las zonas de alta temperatura se minimiza (Seidel et al., 2022).

Las desventajas de estos arreglos de columnas son:

- El diseño de la columna está delimitado al manejo de una presión balanceada, que se alcanza por el diseño del mismo número de etapas en cada lado de la partición. En algún caso donde ocurra espumeo puede llegar a producir un desbalance hidráulico y cambios de las separaciones del vapor de las condiciones de diseño.

- Todo el calor se suministra a la temperatura más alta y todo el calor rechazado a la temperatura más baja de la separación.

Se ha demostrado que las variables del proceso tienen efectos significativos sobre la eficiencia de la energía de una columna de pared divisoria comparándose al efecto de variables estructurales (Villegas-Urbe et al., 2022). Para la operación estable de una columna de pared divisoria, la alimentación debe ser un líquido saturado o subenfriado o se debe manipular la calidad de la alimentación para ajustar la razón de separación de vapor.

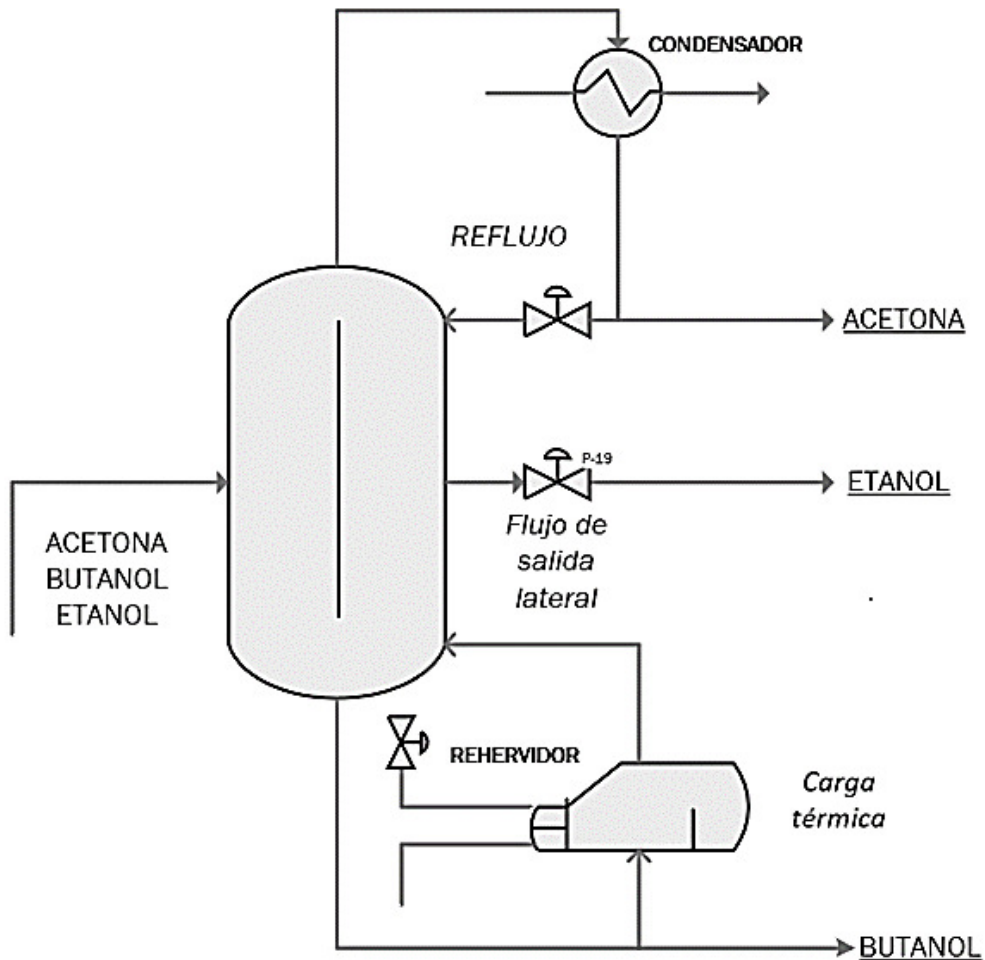


Figura 2.3. Columna de pared divisoria del proceso de obtención de biocombustibles de (Errico et al., 2017).

2.4 MODELADO Y CONTROL

Las columnas de destilación binaria representan un problema difícil e interesante en el diseño de control no lineal debido a la compleja dinámica que rodea al sistema, las interacciones termodinámicas presentes en cada placa, la interacción entre las corrientes internas y externas en el sistema, la no linealidad y las características de distribución de la columna representada por las curvas de composición y temperatura (Lu et al., 2021). Esto se debe a que solo se utiliza una pequeña fracción de la columna para la separación mayor mientras que otras partes son responsables solo del perfeccionamiento de los productos (Shin et al., 2020b).

Se han presentado modelos matemáticos complejos para columnas de destilación binaria, representando la dinámica del proceso real, pero debido a la alta complejidad, se realizan suposiciones como:

- Todos los flujos de entrada y de salida de la torre se encuentran en fase líquida.
- Alimentación en un único plato.
- La columna es adiabática.
- El condensador es total.
- En ocasiones la presión a lo largo de la columna es constante.
- Cada plato tiene una eficiencia del 100%.

Por lo que para fines de modelado y control es difícil su implementación y dado las tendencias actuales se ha optado por otras alternativas. Las redes neuronales pueden modelar el sistema sin las suposiciones que afectarían el resultado al mostrar la dinámica del proceso (El-Gendy et al., 2019; Lu et al., 2021).

Recientemente, (Shin et al., 2020b) propuso un sistema MCP usando redes neuronales implementado en una columna de destilación convencional comparando los resultados con un control PID. Los datos alimentados a la red fueron obtenidos a partir de una simulación de ASPEN Dynamics, la cual tiene las suposiciones enlistadas, para poder optimizar la estructura de la red neuronal usada en el controlador, los datos se pueden observar en Fig. 2.4. Esta determinación de la estructura de la red neuronal artificial se basó en comparar el Error Cuadrático Promedio de redes neuronales con diferente número de neuronas en la capa escondida.

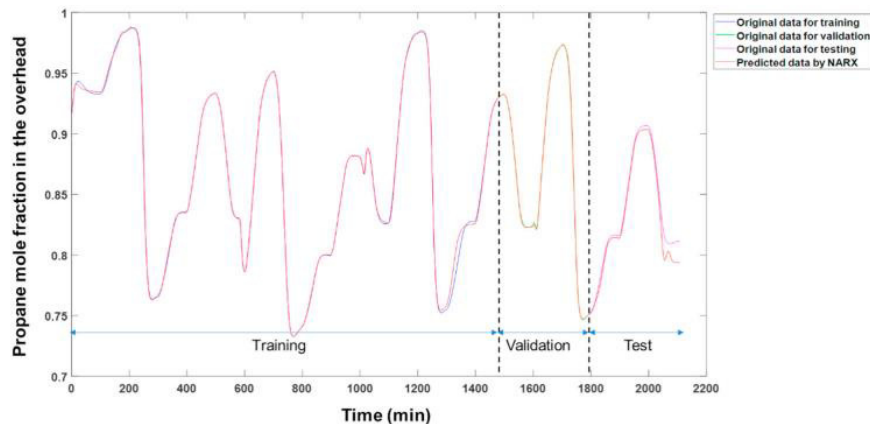


Figura 2.4. Modelado de una columna de destilación convencional con redes neuronales de (Shin et al., 2020b) controlando la pureza de los compuestos a separar.

Finalmente, a los controladores se les determina un setpoint al iniciar la simulación y con una modificación a los primeros 900 minutos de 1800 se observa como el controlador basado en las redes neuronales obtiene un mejor ajuste de la composición del compuesto de interés, en este caso del propano.

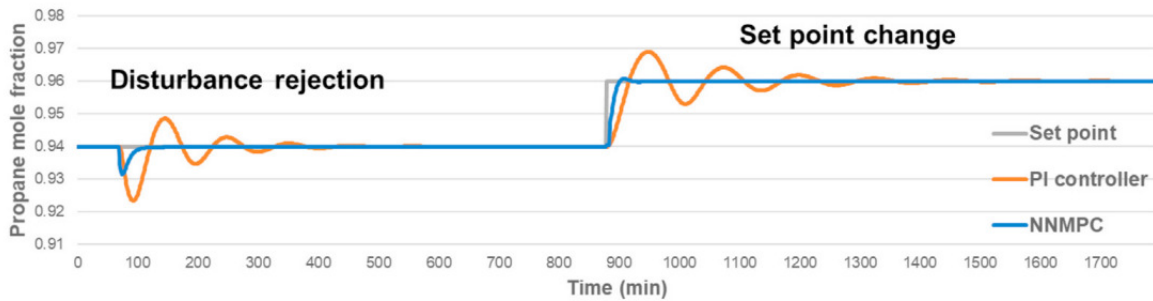


Figura 2.5. Resultados obtenidos de (Shin et al., 2020b).

Para el caso de sistemas más complejos, como en el propuesto por (Neves et al., 2021), se realiza la implementación de un MCP a base de una inteligencia artificial en una columna extractiva para una mezcla donde se cuenta un azeótropo entre el etanol y el agua. Se determina la estructura de la RNA en función del error cuadrático promedio y, por último, se compara la acción del controlador para distintos cambios del setpoint en la composición del etanol purificado.

En los resultados mostrados por (El-Gendy et al., 2019) se muestra como las redes neuronales pueden tener una gran capacidad para controlar una columna de pared divisoria para la separación de la mezcla ternaria etanol-propanol-butanol y en (de Araújo Neto et al., 2021) se muestra la comparación de controlabilidad para una columna extractiva de pared divisoria para cambios de setpoint, donde el controlador a base de redes neuronales es programado en MATLAB en su herramienta de gran capacidad SIMULINK. Este resultado del proceso de Fig. 2.6 es comparable con el de (Shin et al., 2020b) al tener un esquema de control parecido.

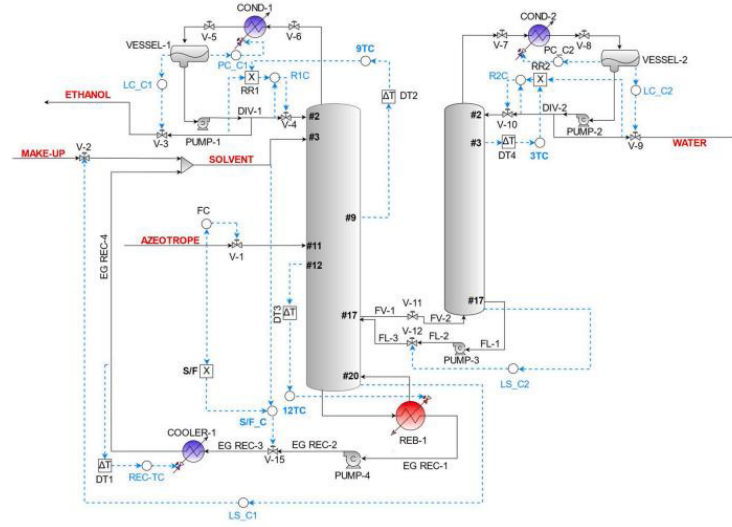


Figura 2.6. Sistema propuesto en (de Araújo Neto et al., 2021).

CAPÍTULO 3

REDES NEURONALES

3.1 REDES NEURONALES

Como se había mencionado la Red Neuronal Artificial (RNA) es una herramienta computacional inspirada en la estructura de una neurona biológica con una gran cantidad de aplicaciones incluyendo clasificación, optimización y predicción debido a su gran capacidad para aprender, almacenar y predecir datos (Yan et al., 2020). Siendo la capa de red el tipo más común de arquitectura de red neuronal constituida por una capa de entrada que representa los datos o información que se le proporciona a la red (Cheng et al., 2021), Fig. 3.1.

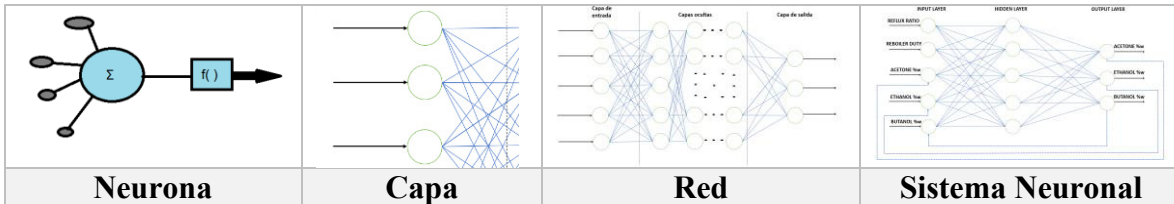


Figura 3.1. Estructura jerárquica de una RNA.

Se denomina procesador elemental o neurona a un dispositivo simple de cálculo que a partir de un vector de entrada procedente del exterior o de otras neuronas, proporciona una única respuesta o salida (Cheng et al., 2021), Fig 3.2. Aunque un modelo es siempre una simplificación de la realidad, por lo que nunca será del todo correcto, pese a ello, algunos modelos nos pueden resultar útiles en la práctica (Ricardo and Rafael, 2015).

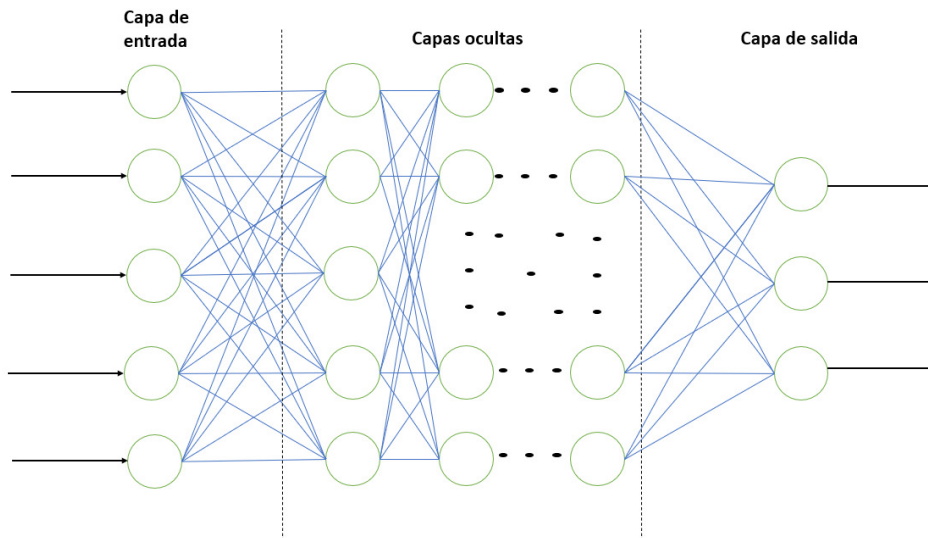


Figura 3.2. Capas de una RNA.

Desde el punto de vista computacional lo único que nos interesa es que la señal proveniente de una entrada afecte el estado de una neurona. Amplificada o atenuada se combina con otras señales de entrada para determinar el nivel de activación de la neurona. A partir de ese nivel

de activación la salida de la neurona puede variar o no (Domínguez Mayorga et al., 2012). Por esto se modela la sinapsis como un número real, conocido normalmente como peso.

Los elementos que constituyen una neurona son los siguientes:

- Conjunto de entradas.
- Pesos sinápticos de la neurona que representan la interacción entre cada neurona presináptica y postsináptica.
- Sesgo (en inglés *bias*).
- Regla de propagación, que proporciona el valor del potencial posináptico de la neurona en función de sus pesos y entradas.
- Función de activación.
- Función de salida.

Las variables de entrada y salida pueden ser binarias o continuas, dependiendo del modelo utilizado y la aplicación del sistema. El rango de los valores que una neurona de salida continua se suele limitar a un intervalo definido como $[-1, +1]$. Las Ecuaciones 1 y 2 representan el modelo equivalente de las conexiones sinápticas en una k neurona, siendo la Ecuación 1 la que describe la regla de propagación que permite obtener a partir de las entradas y los pesos el valor del potencial postsináptico de la neurona. La función más habitual es de tipo lineal y se basa en la suma ponderada con los pesos sinápticos que también puede interpretarse como el producto escalar de los vectores de entrada y pesos (Zhang et al., 1998).

$$u_k = \sum_{j=1}^n w_{kj} x_j \quad (1)$$

$$s_k = \varphi(u_k + b_k) \quad (2)$$

donde el vector x_j es el conjunto de señales de entrada, w_{kj} es el conjunto de pesos sinápticos de la neurona k , u_k es la combinación lineal de las entradas ponderadas, b_k es la polarización y s_k es la señal de salida de la neurona para n número de datos.

La función de activación φ sirve para limitar el rango de salida de la neurona y puede ser lineal o no lineal. Proporciona el estado de activación actual a partir del potencial postsináptico y del propio estado de activación anterior, aunque en muchos modelos de redes neuronales se considera que el estado actual de la neurona no depende de su estado anterior, sino solo del actual. La función de activación se suele considerar determinista y en la mayoría de los modelos es monótona creciente y continua. La más simple de todas es la función identidad o lineal, la cual se puede generalizar al caso de una función lineal cualquiera. En ocasiones los algoritmos de aprendizaje requieren que la función de activación deba cumplir la condición de ser derivable (Y. Liu et al., 2022). En este estudio comparamos el rendimiento de las funciones Tanh, Lineal y ReLU, Fig. 3.3, para la predicción de la pureza de los compuestos donde se tenga manipulación de variables en función del tiempo de operación.

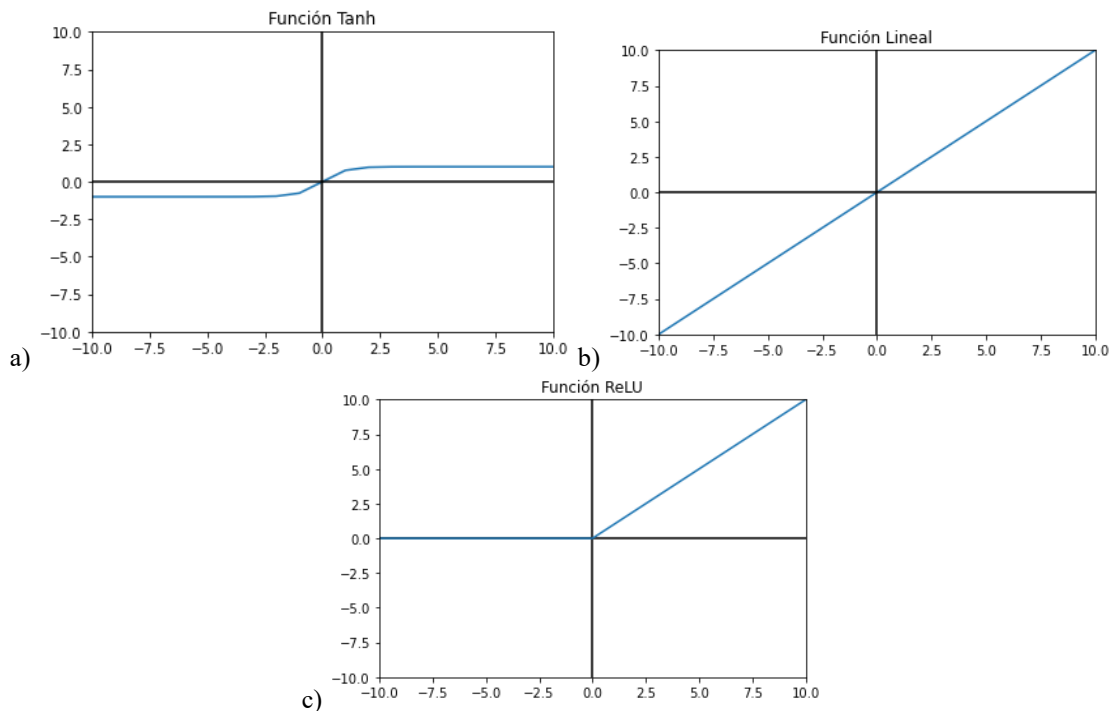


Figura 3.3. Funciones de activación.

El objetivo del entrenamiento de la red neuronal es minimizar la función de pérdida encontrando los pesos adecuados, asegurando una buena generalización. Para determinar estos pesos se lleva a cabo un algoritmo numérico. Por lo que el optimizador es el encargado de generar pesos cada vez mejores al calcular el gradiente de la función de pérdida por cada peso de la red. El conjunto de métodos iterativos para la reducción de la función de pérdida es conocido como los métodos de optimización basados en el gradiente descendiente (Nguyen and Nguyen, 2022). Eligiendo la topología de la red neuronal a partir de la mostrada en (Shin et al., 2020b).

Dentro de la práctica se hace necesario el disminuir gradualmente la tasa de aprendizaje utilizada por el gradiente descendente estocástico. Dada la importancia clave que tiene la tasa de aprendizaje en los resultados que se obtienen al aplicar un algoritmo de optimización para entrenar una red neuronal artificial, no debería sorprendernos demasiado que se hayan propuesto numerosos métodos para ajustar automáticamente las tasas de aprendizaje que gobiernan el proceso de aprendizaje de una red neuronal.

Adaptive Moment Estimation (ADAM). Este algoritmo combina estrategias utilizadas en optimizadores como AdaGrad y RMSPROP. Estos algoritmos están constituidos en base a sus predecesores. Utiliza una media móvil exponencial de los gradientes para ajustar las tasas de aprendizaje (Yadav and Anubhav, 2020).

Root Mean Square Propagation (RMSPROP). La idea de RMSPROP es combinar la robustez de Rprop, la eficiencia del aprendizaje estocástico con minilotes y el promediado de gradientes sobre minilotes. En este algoritmo de optimización se mantiene un factor de entrenamiento diferente para cada dimensión, pero en este caso el escalado del factor de

entrenamiento se realiza dividiéndolo por la media del declive exponencial del cuadrado de los gradientes (Lee et al., 2021).

Stochastic Gradient Descent (SGD). Uno de los métodos de optimización para el cálculo de la derivada parcial de la función de pérdida consiste en la introducción de un comportamiento estocástico (aleatorio). Limitando el cálculo de la derivada a tan solo una observación y no varias (Yan et al., 2020).

Para obtener una buena minimización de la función de pérdida, se comparará la elección del optimizador a partir de una función de activación definida para obtener una buena predicción de las series temporales gracias a su velocidad de convergencia y su velocidad de generalización (Manickam et al., 2021). Los optimizadores comparados son el Adaptive Moment Estimation (ADAM), el Root Mean Square Propagation (RMSPROP) y el Stochastic Gradient Descent (SGD).

3.2 ARQUITECTURA DE LAS RNA

Al analizar con detalle los modelos más utilizados de neuronas artificiales, es posible combinar colecciones de neuronas para resolver complejos problemas de interés. El perceptrón creado por Frank Rosenblatt es el modelo más simple y antiguo de una neurona biológica en una red neuronal artificial, cuyo funcionamiento consiste en tomar unas entradas, sumarlas, aplicar una función de activación y pasarlas a la capa de salida.

3.3 REDES FEED-FORWARD

La red simple, con una única capa, es el caso más simple de red neuronal donde las neuronas de la capa de entrada se limitan a recibir las señales de entrada provenientes del exterior distribuyéndolas a las neuronas de la capa de salida que es la única capa de la red que tiene una función.

Redes multicapa. Al añadir capas intermedias en una red neuronal simple, las capas de salida y de entrada ya no serán visibles desde el exterior, lo que nos obligará a utilizar algoritmos como *backpropagation* para ajustar los parámetros internos con la finalidad de obtener un sesgo menor (Tian et al., 2021).

Las redes neuronales actuales suelen incluir múltiples capas ocultas para aprovechar el potencial del aprendizaje profundo cuando se diseñan soluciones modulares para problemas complejos.

Las funciones de activación, como la tangente hiperbólica, pueden resultar útiles en situaciones en las que un cambio de signo en la salida de la neurona influya en el comportamiento de las neuronas de las siguientes capas. Si la salida bipolar de una neurona se conecta a la entrada de otra con un peso sináptico positivo, la conexión será excitatoria cuando la salida sea positiva e inhibitoria cuando sea negativa o al revés si el peso sináptico es negativo (Tian et al., 2021).

3.4 REDES RECURRENTE

Las redes recurrentes son la arquitectura que se utilizan en mayor proporción en la actualidad. Este tipo de redes fueron las que recibieron más atención por parte de los investigadores por sus características en cuanto a tiempo de procesamiento, ya que hacían viables las simulaciones con los equipos computacionales de la época.

Otra arquitectura de red neuronal ampliamente utilizada en la actualidad es la red de neuronas convolucionales (CNN), empleada en el reconocimiento y procesamiento de imágenes. Esta arquitectura fue específicamente diseñada para el procesamiento de la información contenida en los píxeles de una imagen, aunque posteriormente se ha ampliado su campo de aplicación a datos que tengan forma matricial. Las CNN hacen uso de tres tipos distintos de capas, además de la capa de entrada: convolucionales, de agrupación y capas completamente conectadas (Aslan et al., 2022). La capa de entrada contiene los valores de los píxeles de la imagen, las capas convolucionales procesan los datos de entrada que extraen características de la imagen, estas capas reducen la dimensionalidad de los datos eliminando la redundancia innecesaria y las capas completamente conectadas.

3.5 RED NEURONAL TIPO LSTM

La red tipo LSTM (*Long Short-Term Memory*) es una variante de las redes neuronales recurrentes, que se distinguen por tener al menos uno o más bucles de retroalimentación que tienen un profundo impacto en la capacidad de aprendizaje de la red neuronal; en este caso la red LSTM tiene la capacidad de aprender información dependiente a largo plazo, lo que supone un avance considerable en problemas relacionados con el análisis de series temporales. Este tipo de redes para poder realizar buenas predicciones necesitan una mayor potencia computacional ya que como se muestra en la Fig. 3.4 tiene una retroalimentación basada en un paso de tiempo, en este estudio se utiliza un paso de 5 para la retroalimentación, gracias a la memoria que funciona mediante un algoritmo basado en una puerta de olvido, una puerta de entrada y una puerta de salida con la que es posible tener una buena predicción para una gran cantidad de datos (Hochreiter and Schmidhuber, 1997).

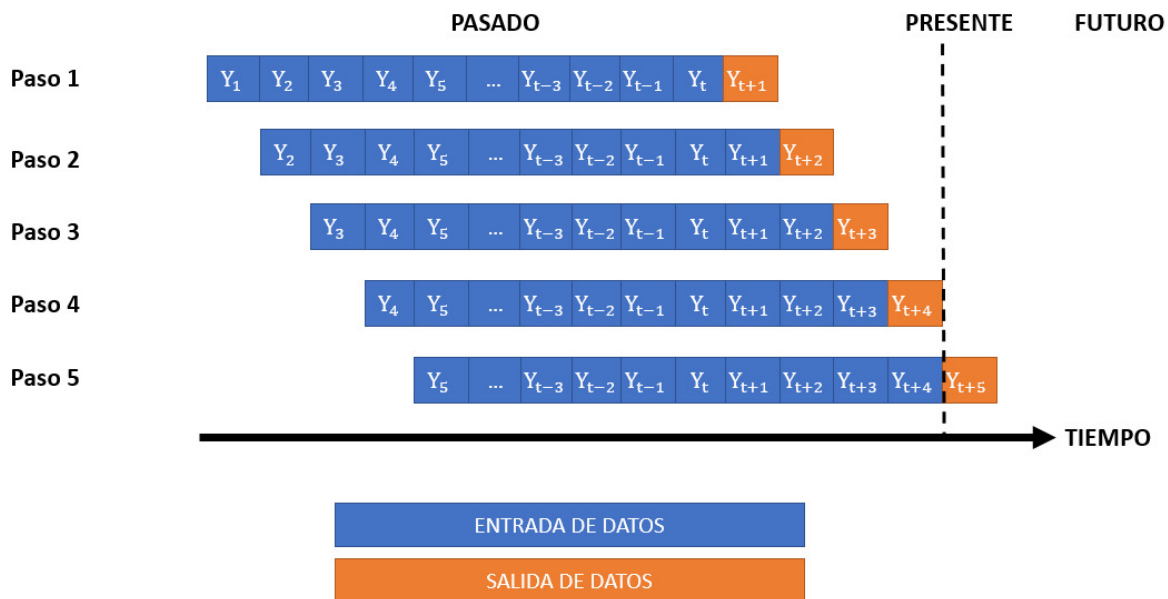


Figura 3.4. Estructura de una red tipo LSTM.

Las redes LSTM son capaces de añadir o desechar la información que se considere relevante para el procesamiento de la secuencia. Comparada con una celda de red recurrente básica; la celda LSTM tiene una entrada y salida adicional, este elemento adicional se conoce como celda de estado. La celda de estado es como una banda transportadora a la que se pueden añadir o donde se pueden remover datos que no se desean preservar en la memoria de la red. Para añadir o remover datos se utilizan varias compuertas como compuerta de olvido: que permite eliminar elementos de la memoria; compuerta de entrada: que permite añadir nuevos elementos de la memoria; compuerta de salida: que permite crear el estado oculto actualizado, Fig. 3.5. Tienen una función similar a una válvula donde totalmente abiertas permiten el paso de la información y totalmente cerradas la bloquean por completo.

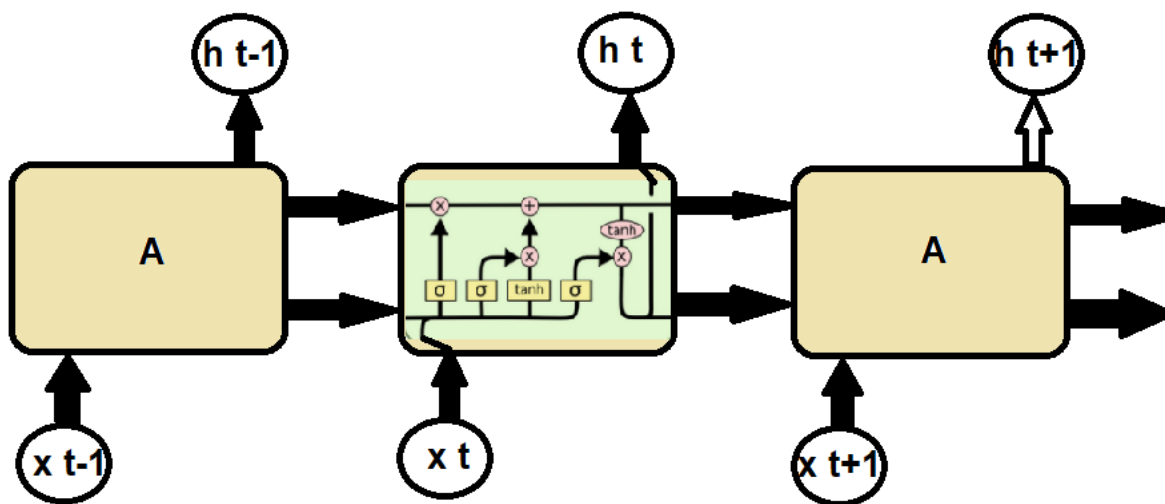


Figura 3.5. Estructura secuencial de una neurona tipo LSTM.

Se debe decidir qué información se debe mantener y cual borrarse, usando de analogía la capacidad que tiene el ser humano de olvidar y memorizar información. Usando una función de activación determinada, en este caso la función sigmoide, se transforman los vectores del output del módulo anterior y el input que se está procesando en el módulo actual. Un valor de 1 indica que se mantiene ese valor mientras que un 0 indica que ese valor es prescindible y se debe olvidar.

Una vez el módulo ha olvidado la información que ha interpretado como necesaria o no, se procede a memorizar la nueva información que se considera relevante. Partiendo del mismo input que en el paso anterior, el objetivo es generar el nuevo estado del módulo actual a partir de dos pasos.

Una vez hemos preparado el estado de la red actual, ya solo queda preparar la salida del módulo actual y pasar toda la información necesaria al siguiente. Para generar la nueva salida se toma en cuenta el estado de la red después de someterla a una función tanh y posteriormente se decide en qué medida se usan estos valores con un vector cuyo resultado es obtenido al someter los valores a una función sigmoide.

Este tipo de redes neuronales son aplicables en aquellos problemas donde se tienen una gran cantidad de datos y el contexto de corto o largo plazo de estos tiene un impacto relevante en los resultados. Es por eso por lo que, se aplica a series de datos en que éstos siguen algún tipo de patrón o reglas, generalmente con combinaciones complejas de información de varios tipos, que son las que la arquitectura LSTM va a aprender y que van más allá de unas simples reglas gramaticales como podíamos tener en otro tipo de arquitecturas. Aunque normalmente son aplicados a problemas de predicción de series temporales, (Kumari and Toshniwal, 2021), también son aplicables a otro tipo de magnitudes, por ejemplo, dimensiones (Khodayar and Wang, 2019) y económicas (Chang et al., 2019).

CAPÍTULO 4

METODOLOGÍA

4.1 METODOLOGÍA

La determinación del número de capas ocultas y el número de neuronas de cada capa, para una RNA como la que se observa en la Fig. 4.1, es un reto en la construcción de la arquitectura óptima del modelo de la red neuronal que se puede llevar a cabo minimizando, en este caso, el MSE (Error cuadrático promedio entre los datos del simulador y de la RNA) con respecto a la variación del número de neuronas ocultas (Shin et al., 2020a). Este valor estadístico es calculado usando el número total de datos (n), el valor obtenido de la simulación (y_i) y el valor predicho (\bar{y}_i), como se observa en la Ecuación 3.

$$\text{MSE} = \frac{\sum_{i=1}^n (y_i - \bar{y}_i)^2}{n} \quad (3)$$

La cual es la medida de error más utilizada para estimar la calidad de un modelo de predicción cuantitativa, que se aplica sobre las n muestras del conjunto de datos sobre el que se está estimando la calidad del modelo cuantitativo. En ocasiones no es necesario calcular la media, sino que nos valdrá la suma de errores al cuadrado (SSE), (Shin et al., 2020a).

$$\text{SEE} = n * \text{MSE} = \sum_{i=1}^n (y_i - \bar{y}_i)^2 \quad (4)$$

Dada una colección de modelos para un conjunto de datos, el valor del AIC (criterio de información de Akaike) estima la calidad de cada modelo, en relación con cada uno de los otros modelos (Portet, 2020), como se observa en la Ecuación 5 a partir del número de datos (n) y el error cuadrático promedio (MSE). Por lo tanto, el AIC proporciona un medio para la selección de modelos. Se ocupa del equilibrio entre la bondad del ajuste del modelo y la complejidad de este. El valor obtenido no proporciona una prueba de un modelo en el sentido de probar una hipótesis nula, por lo que no puede decir nada sobre la calidad del modelo en un sentido absoluto. Si todos los modelos candidatos se ajustan mal, el AIC no avisará de ello. La fórmula del AIC depende del modelo estadístico. Un valor de AIC más bajo significa que un modelo determinado describe mejor los datos que otros modelos con valores más altos.

$$\text{AIC} = n * \log(\text{MSE}) + \frac{2 * m * n}{n - m - 1} \quad (5)$$

La normalización de los datos es un paso importante en el entrenamiento de una red neuronal, (Han and Kang, 2022), ya que los datos obtenidos para la columna intensificada del calor del reboiler se encuentran en el orden de 1×10^{-4} y los datos obtenidos de la pureza en el orden de 1×10^{-1} por lo que se transforman todos los valores de las características en un intervalo de $[0,1]$; matemáticamente hablando y_i se refiere a los valores de los datos, $\min(y_i)$ es el valor mínimo y, $\max(y_i)$ se refiere al valor máximo, como se observa en la Ecuación 6.

$$y_{\text{scaled}} = \frac{y_i - \min(y_i)}{\max(y_i) - \min(y_i)} \quad (6)$$

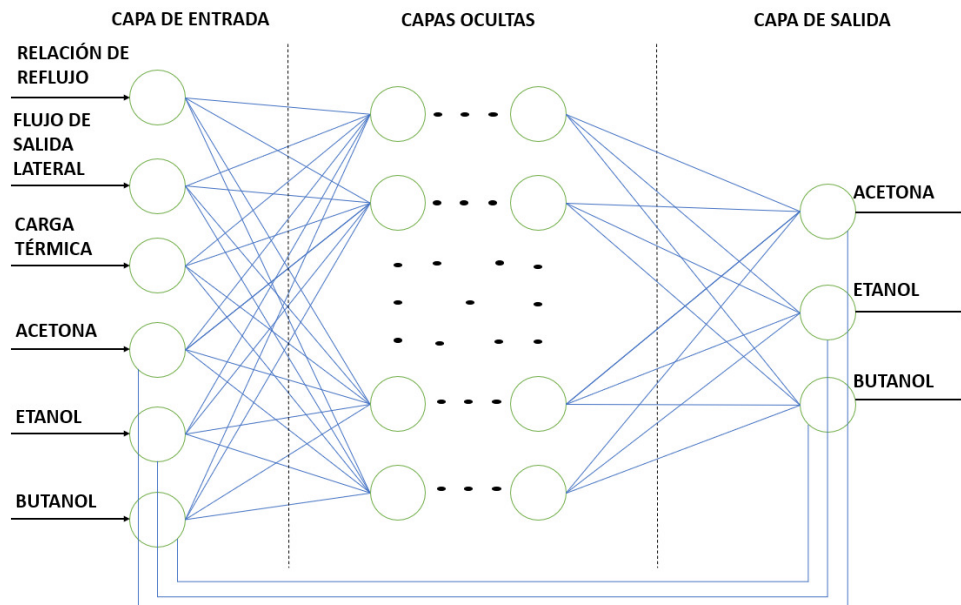


Figura 4.1. Estructura para el caso del sistema intensificado.

La época en una red neuronal es el número de veces que se van a pasar cada grupo de datos de entrenamiento por la red. Los datos de entrada son los obtenidos por el simulador de las variables manipulables y las variables perturbadas obteniendo como datos de salida la predicción las variables perturbadas. Los datos generados son alimentados a una red neuronal de una sola capa oculta para la determinación del número de épocas los cuales se subdividieron en un 24% para el entrenamiento y un 76% para la prueba de ambos sistemas analizados, Fig. 4.2. En la Fig. 4.3 se puede observar que aumentar el número de épocas hasta 3000 el error cuadrático promedio disminuye, después de este valor obtenido de la desviación estándar de los 100 entrenamientos el error aumenta, por lo que se usa este valor como el número de épocas (Han and Kang, 2022).

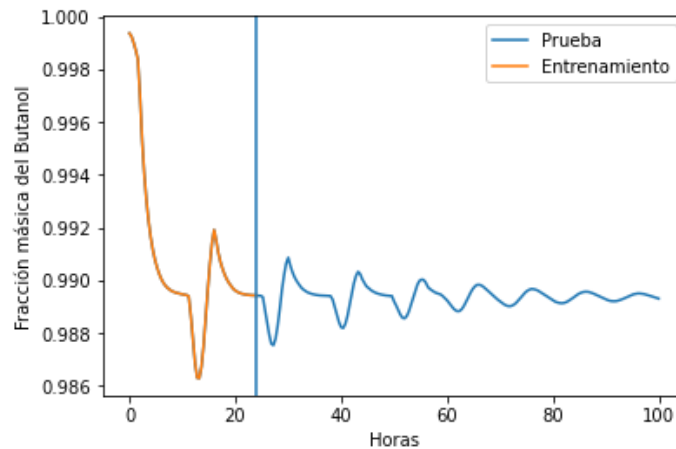


Figura 4.2. Ejemplo de división de datos obtenidos del perfil de composición del butanol para prueba y entrenamiento en la columna intensificada.

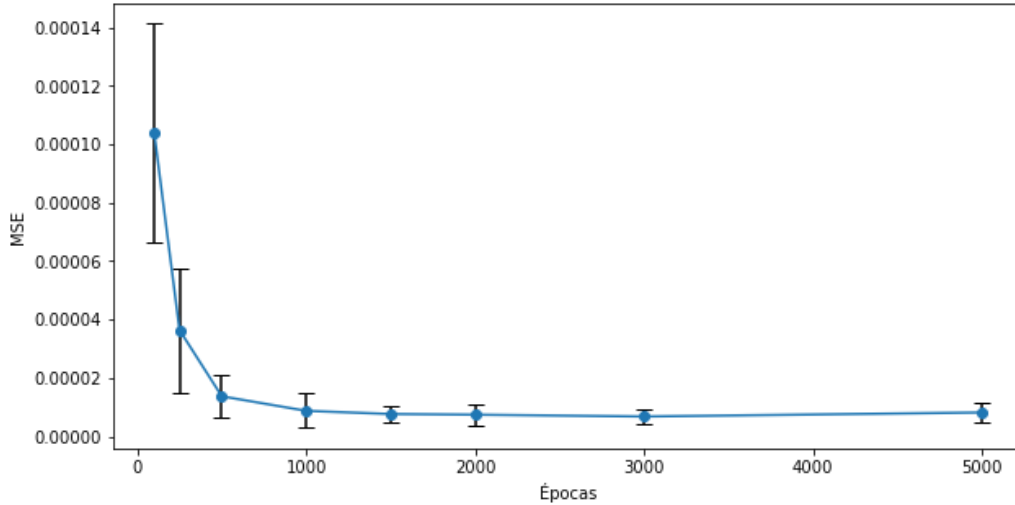


Figura 4.3. Determinación del número de épocas utilizadas.

4.2 OPTIMIZACIÓN ESTRUCTURAL

Con la estructura de la RNA descrita en el Capítulo 3, se realiza el análisis para cada caso donde se altere la configuración, complejidad y las variables alimentadas a la red. Como se observa en la Fig. 4.4 se inicia la obtención de resultados para el número de neuronas ocultas de una sola capa oculta a la cual se le modifica la función de activación para poder comparar y determinar el que mejor prediga los datos de prueba. El cambio de las variables manipulables se realiza omitiendo de la capa de entrada una de las 3 variables, por lo que solo se alimentaría las 2 variables restantes.

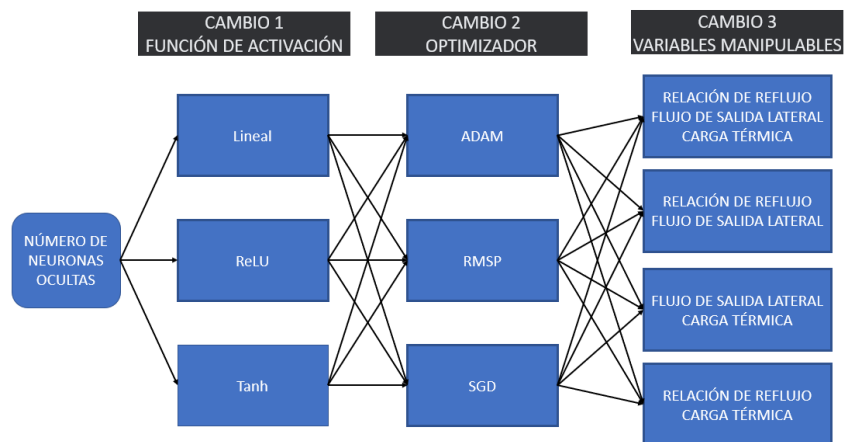


Figura 4.4. Cambios en la estructura de la RNA para poder obtener el mejor modelo en el caso intensificado.

4.3 NÚMERO DE CAPAS OCULTAS.

Una vez que se ha definido la función de activación, el optimizador y las variables con las que se determina el modelo que mejor describa la dinámica del proceso se procede a aumentar el número de capas de neuronas ocultas a partir del número de neuronas elegidas por el menor valor del AIC, Ecuación 5. Así se puede observar el efecto que se tiene al aumentar el número de capas y neuronas en la predicción de las fracciones másicas de los compuestos purificados. Todas las neuronas que se agreguen a la red tendrán el mismo funcionamiento en el sistema al ser todas del tipo LSTM. Este aumento también se tomará en el criterio de selección al aumentar la complejidad de la red y por consiguiente del modelo.

4.4 PREDICCIÓN

Una vez obtenidos los valores que describen cuantitativamente la capacidad de predicción de la red neuronal, se lleva a cabo un ensayo donde una estructura definida de la red realiza la predicción de los perfiles de composición obteniendo de esta forma la capacidad de analizar cualitativamente los resultados para poder colaborar el funcionamiento correcto del modelo cuando describe la dinámica del proceso en el cual se está usando esta herramienta. El análisis cualitativo se realiza de esta forma ya que una predicción con un bajo valor del MSE puede tener un sobreajuste que sea la causa del bajo valor, es por eso por lo que se comparan los perfiles de composición obtenidos por el simulador y el obtenido por la predicción de la red neuronal artificial.

4.5 CASO DE ESTUDIO

Se han considerado un proceso híbrido que incluye una columna de extracción líquido-líquido, y una columna de pared dividida. Este diseño está compuesto por una columna de extracción líquido-líquido (Fig. 4.5) en la que se utiliza acetato de n-hexilo como agente extractor para romper azeótropos tanto homogéneos como heterogéneos y una columna de pared dividida (ver Fig. 2.3).

La obtención del biobutanol puede darse a través del proceso de fermentación ABE a partir de biomasa como la lignocelulosa (Pereira et al., 2015), en este proceso se obtiene una mezcla de acetona, etanol y butanol, la cual crea un reto al tratar de reducir la energía necesaria en el proceso de separación (González-Bravo et al., 2016). Este compuesto al ser utilizado como combustible es llamado biobutanol esto con el propósito de remarcar su origen vegetal, puesto que el butanol puede producirse también a partir de combustibles fósiles, con las mismas propiedades químicas.

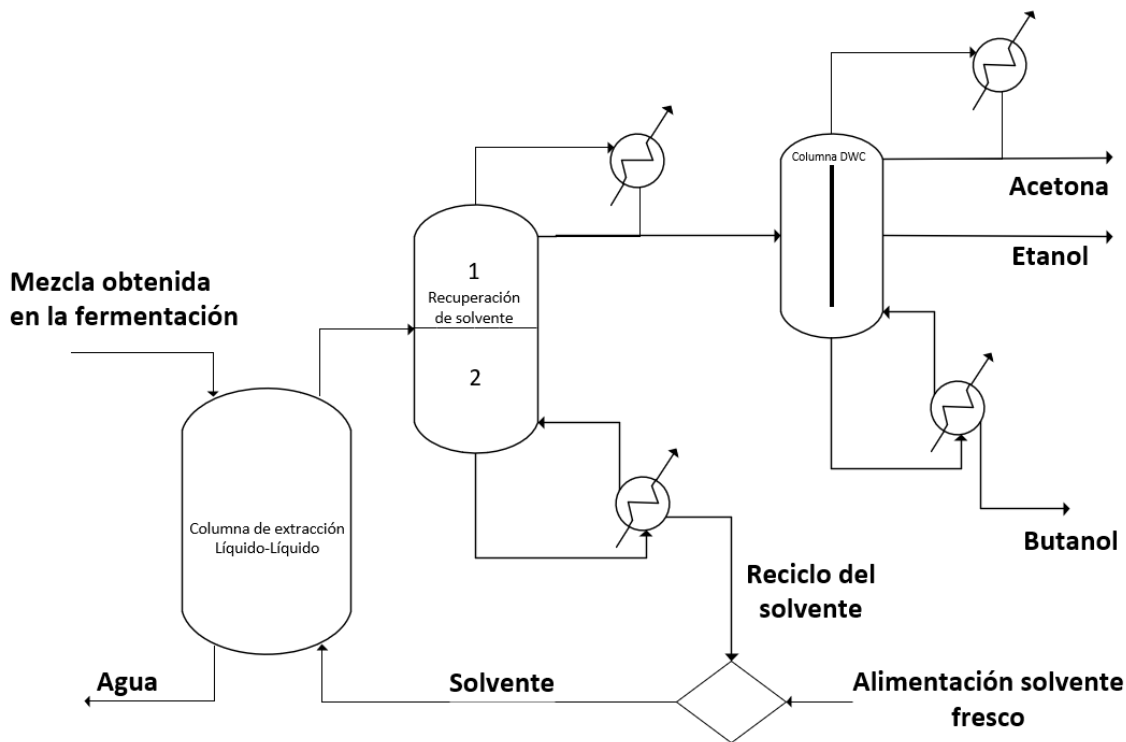


Figura 4.5. Caso de estudio.

Antes de la prueba en ambiente Python 3.8, este diseño se simuló primero en Aspen Dynamics, como se describe en la metodología para el desarrollo de un modelo de control predictivo para el caso de estudio de una columna de destilación (Shin et al., 2020b), la composición de la alimentación y los parámetros físicos se muestran en la Tabla 4.1 de acuerdo con (Wu et al., 2007).

Tabla 4.1. Flujo de alimentación al sistema de purificación.

Temperatura (°C)	50
Fracción de vapor	0
Flujo (kg h⁻¹)	45,3592
Composición (% mol)	
Biobutanol	0,1128
Acetona	0,0808
Etanol	0,0043
Agua	0,80198

Se eligió el modelo termodinámico NRTL-HOC para modelar la interacción entre todos los componentes (Van der Merwe et al., 2013). Se eligió el acetato de N-hexilo como extractor en la columna de extracción líquido-líquido. La pureza de la acetona se fijó en 0,996 (%wt), la del butanol en 0,995 (%wt) y la del etanol en 0,95 (%wt). El coste de capital y el impacto medioambiental medido a través del ecoindicador 99 obtenido según los resultados de (Errico et al., 2016). La siguiente tabla muestra los parámetros de diseño para la simulación del caso de estudio, siendo C₁ la columna de recuperación del solvente:

Tabla 4.2. Parámetros de diseño para la secuencia considerada como caso de estudio.

Parámetro	Caso de estudio		
	Extractor	C ₁	DWC
Número de etapas	5	23	48
Número de etapas a través de la pared divisoria	---	---	18
Localización de la alimentación	---	12	31
Etapas del flujo de salida lateral	---	---	33
Relación de reflujo	---	0.894	20.314
Flujo del destilado [kg h ⁻¹]	---	21.685	7.697
Flujo de la salida lateral [kg h ⁻¹]	---		0.327
Flujo de separación de líquidos [kg h ⁻¹]	---	---	2.183
Flujo de separación de vapor [kg h ⁻¹]	---		9.821
Flujo del extractante [kg h ⁻¹]	733.873	---	---
Flujo del solvente [kg h ⁻¹]	712.147	--	--
Diámetro [m]	0.335	0.288	0.302
Presión [kPa]	101.3	101.3	101.3
Consumo del condensador [kW]	---	7.241	23.452
Calor del reboiler [kW]	---	66.218	23.818
TAC [k\$ yr ⁻¹]		111.86	
EI99 [puntos k yr ⁻¹]		17.50	
CN		10.35	

Al realizar la simulación en ASPEN Dynamics se obtuvieron una serie de datos y se graficaron para observar la respuesta dinámica ante perturbaciones. Las variables perturbadas; la fracción masa de Acetona, Etanol y Butanol se muestran en la Fig. 4.6 a) y las variables manipulables; relación de reflujo, flujo de salida lateral y carga térmica, se muestran en la Fig. 4.6 b), estos datos se obtuvieron realizando perturbaciones a las variables manipulables, cambiando su set point cada 5 horas hasta llegar a un periodo de análisis de 100 horas.

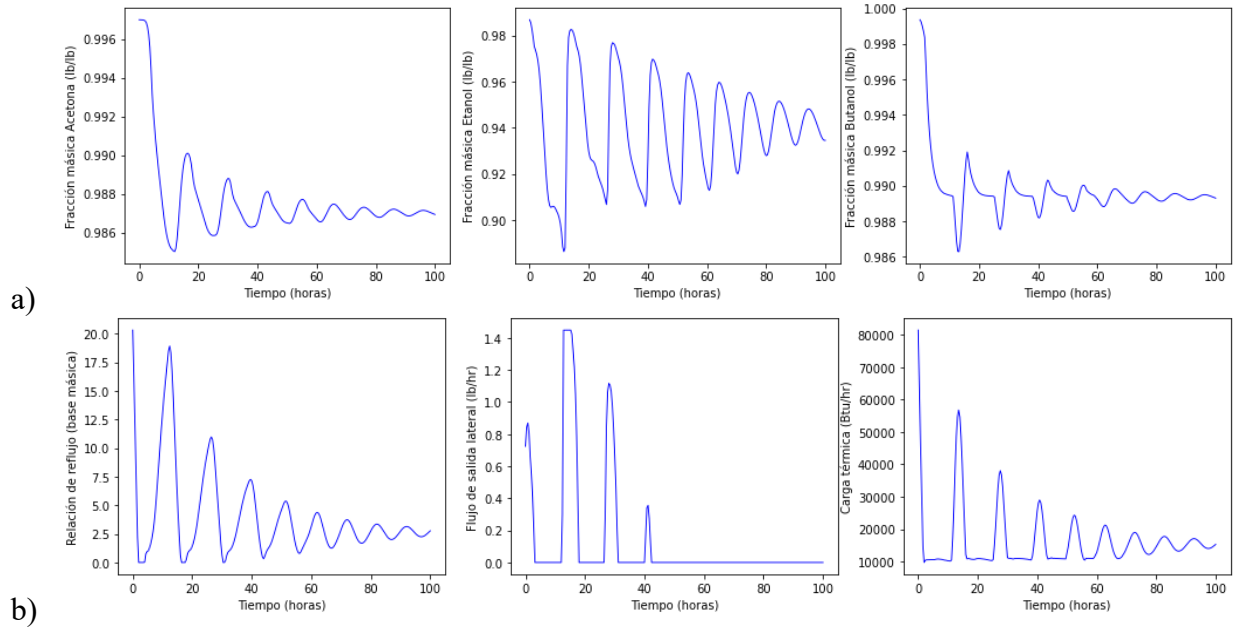


Figura 4.6. Generación de datos del sistema intensificado: a) Variables Manipulables y b) Variables Perturbadas.

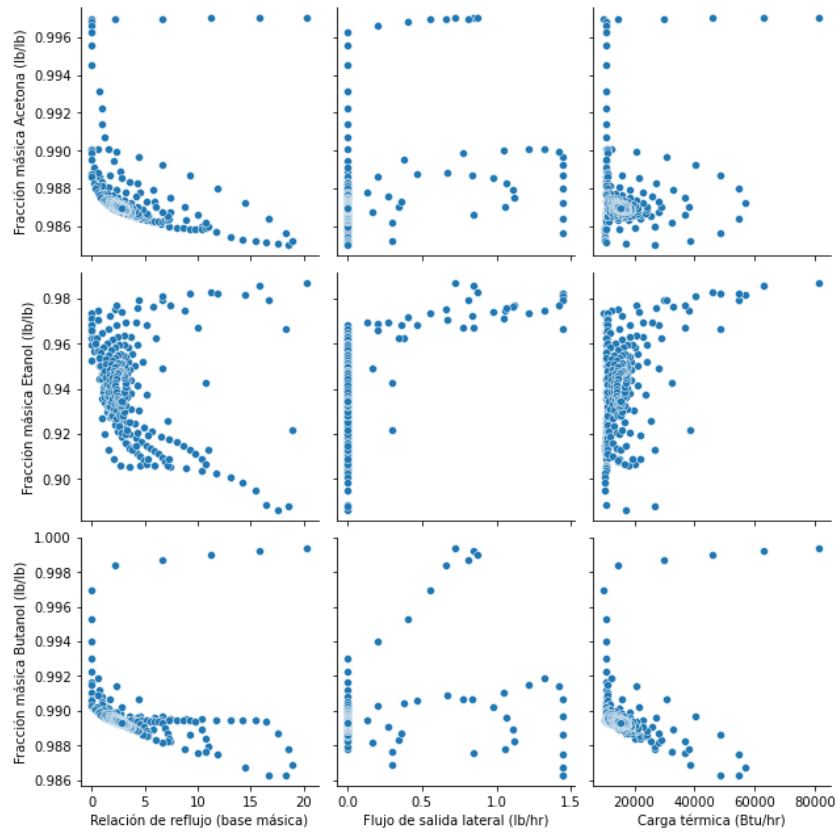


Figura 4.7. Matriz de cofactores de los datos obtenidos de la columna de pared divisoria.

Los datos obtenidos en ASPEN Dynamics fueron alimentados a través del paquete de la librería SEABORN donde fue posible analizar la gran cantidad de datos en la matriz de cofactores, los resultados se muestran en Fig. 4.7. Observando que los datos generados tienen un cierto comportamiento relacionado con las funciones que describen como algunas variables manipulables tienen más impacto en el perfil de composición del butanol, en este caso la relación de reflujo y la carga térmica son las que tienen un comportamiento similar cuando se relacionan con el compuesto ya que la salida lateral crea un comportamiento más aleatorio.

CAPÍTULO 5

RESULTADOS

5.1 RESULTADOS

Se analiza la columna intensificada con 100 iteraciones del entrenamiento de la red neuronal para determinar el valor del MSE y un valor promedio del AIC con sus respectivas barras de error construidas a partir del valor de la desviación estándar. Se inicia el análisis con una RNA de una sola capa oculta de neuronas donde se alimentan las variables manipulables (relación de reflujo, flujo de salida lateral y carga térmica) y las variables perturbadas (fracción másica de la acetona, etanol y butanol). Se predice el perfil de composición de los tres componentes purificados con la red que obtenga el valor del AIC menor.

Para el sistema intensificado se realizó, inicialmente, el análisis de la capacidad de predicción con las diferentes funciones de activación y los diferentes optimizadores para poder determinar la estructura del mejor modelo. En la Fig. 5.1 se puede mostrar la comparación entre las tres funciones de activación para el optimizador RMSPROP donde se observa un valor mayor del error con la función RELU y un comportamiento parecido de las dos funciones restantes.

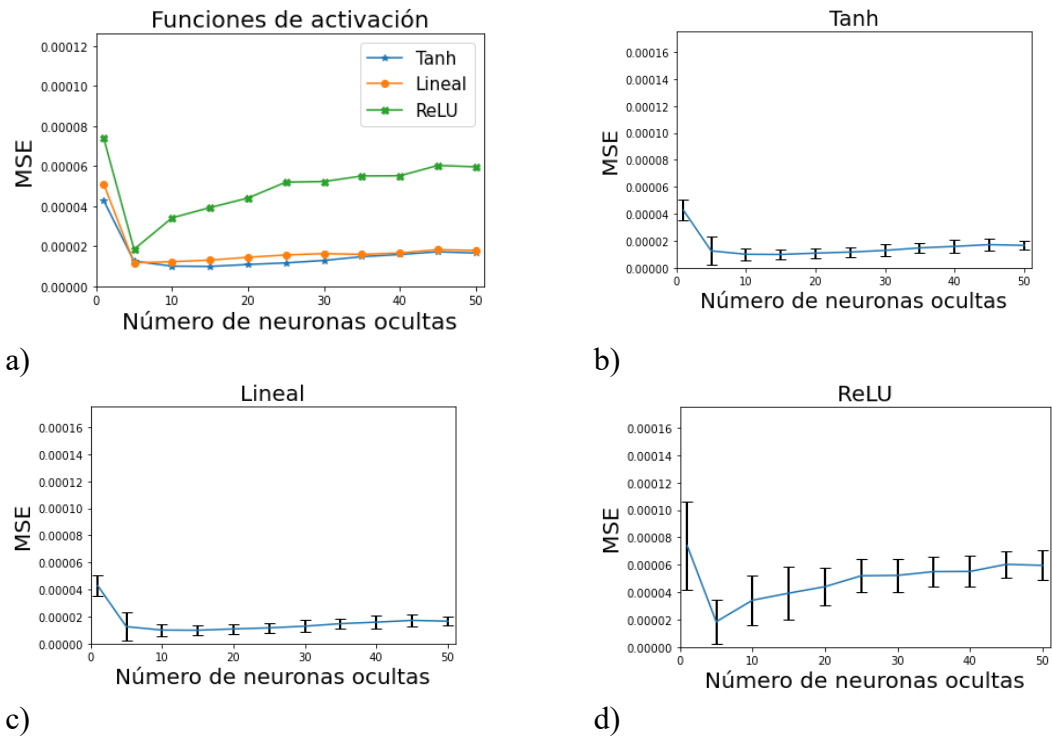


Figura 5.1. Resultados de la capacidad de predicción con diferentes funciones de activación usando el optimizador RMSPROP para el sistema intensificado: a) Comparación, b) Función Tanh, c) Lineal y d) ReLU.

En la Fig. 5.2 se muestra el análisis del AIC con las diferentes funciones de activación para el optimizador RMSPROP, donde en la Fig. 5.2c, que la con la función de activación lineal se obtiene el mejor modelo con 5 neuronas.

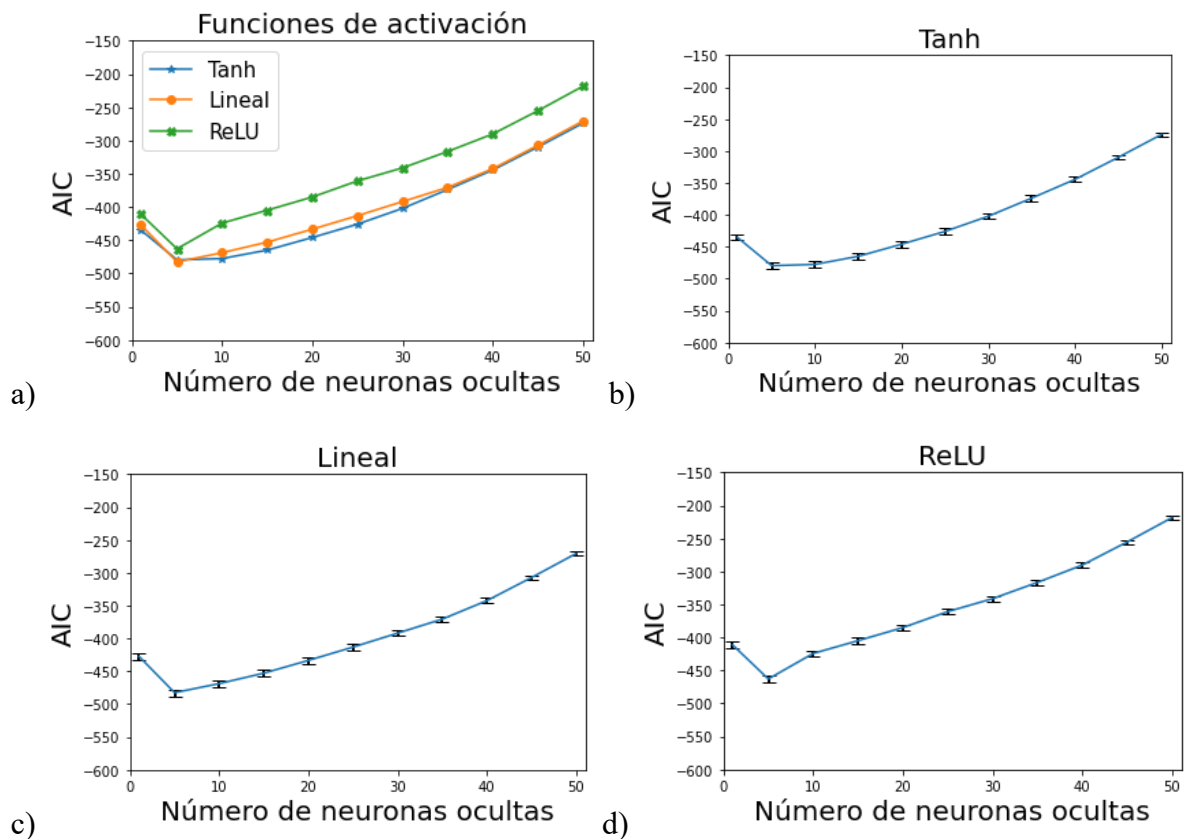
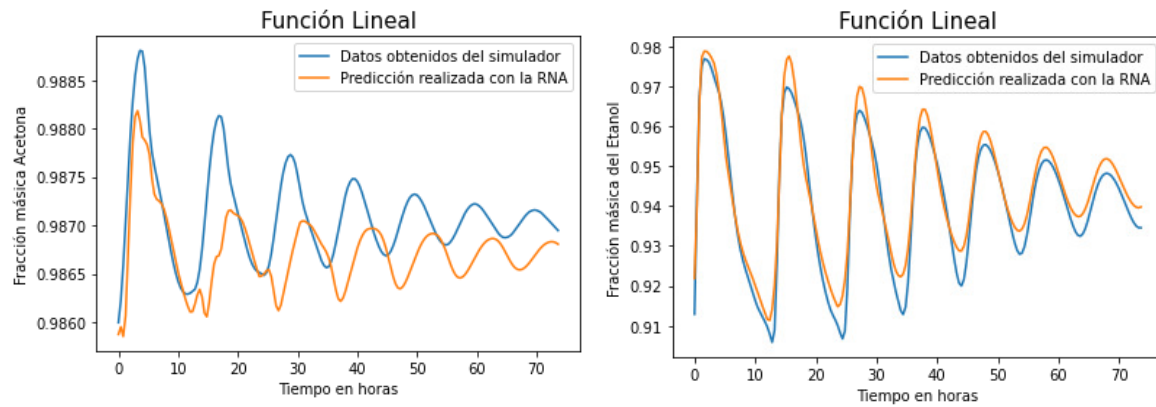


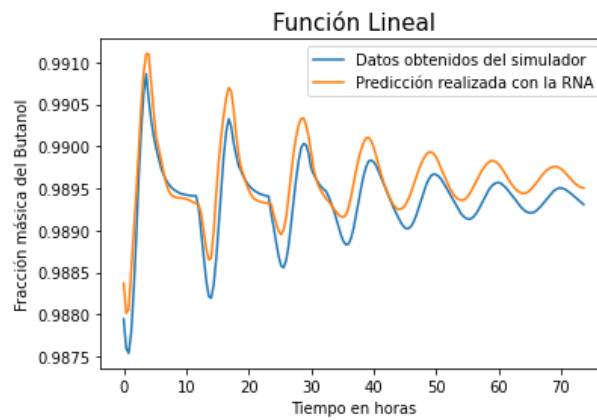
Figura 5.2. Valor obtenido de AIC con diferentes funciones de activación usando el optimizador RMSPROP para el sistema intensificado: a) Comparación, b) Función Tanh, c) Lineal y d) ReLU.

Con el optimizador RMSPROP se analizó la capacidad de predicción con la función de activación lineal y 5 neuronas. Esto se demuestra en la Fig. 5.3 graficando los datos reales (línea azul) y los datos predichos por la red neuronal (línea naranja) de los perfiles de composición de la acetona, el etanol y el butanol.



a)

b)



c)

Figura 5.3. Resultados de la predicción realizada para el sistema intensificado usando la función Lineal, el optimizador RMSPROP y 5 neuronas en la capa oculta: a) Acetona, b) Etanol y c) Butanol.

En este caso se obtiene una predicción de los perfiles del etanol y del butanol con un comportamiento similar a los obtenidos del simulador, mientras que en el caso de la acetona no se logra acercarse a los valores reales.

5.2 ANÁLISIS DE RESULTADOS DE LA RNA CON UNA SOLA CAPA OCULTA

Como se observa en la Fig. 5.1a el valor del MSE alcanza su máximo cuando se tiene una neurona en la capa oculta y conforme aumenta el valor se acerca a un valor constante, reduciendo el tamaño de las barras de error por lo que se entiende que disminuye la variación de los valores obtenidos en las simulaciones realizadas. Utilizando el valor del AIC se determina que para todos los optimizadores el número óptimo de neuronas ocultas es de 5 obteniendo valores del MSE con una magnitud de 1×10^{-5} . El optimizador SGD presenta errores más altos a comparación de los demás, en particular cuando se utiliza este optimizador con la función lineal las predicciones son pobres y con un alto valor del error. Mientras que el optimizador RMSPROP al implementarse con la función lineal y 5 neuronas en la capa oculta presenta el menor valor del AIC del análisis (-475) y el menor valor del MSE (1.04899×10^{-5}), como se observa en la Tabla 5.1.

Tabla 5.1 Comparación de optimizadores en las estructuras con el valor de AIC más bajo obtenido para la columna intensificada.

Optimizador	Función de activación	Neuronas ocultas	MSE ($\times 10^{-5}$)
ADAM	Lineal	5	1.16944
RMSPROP	Lineal	5	1.04899
SGD	ReLU	5	2.72182

5.3 IDENTIFICACIÓN DE VARIABLES CON MÁS IMPORTANCIA-PESO EN EL MODELO DE LA RNA

En esta sección se presenta el resultado de reducir el número de variables alimentadas a la capa de entrada de la RNA con el fin de reducir la variación del error. En este caso las variables manipulables pueden ser reducidas manteniendo en la capa de salida la respuesta del perfil de composición de los compuestos purificados. En la Fig. 5.4 y 5.5 se puede observar que la línea de color azul corresponde a la estructura definida en la sección anterior y las demás líneas corresponden a los modelos donde se solo se alimentan 2 variables manipulables.

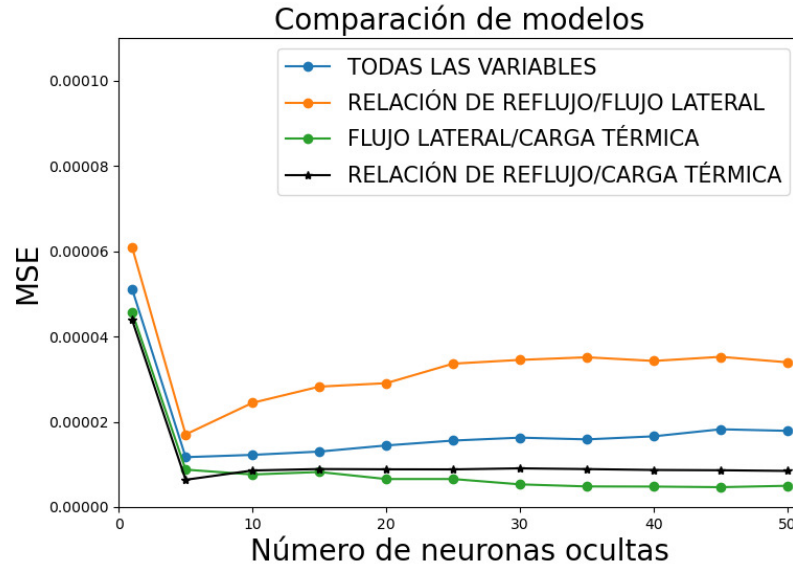


Figura 5.4 Resultados de la capacidad de predicción con diferentes funciones de activación usando el optimizador de RMSPROP para el sistema intensificado.

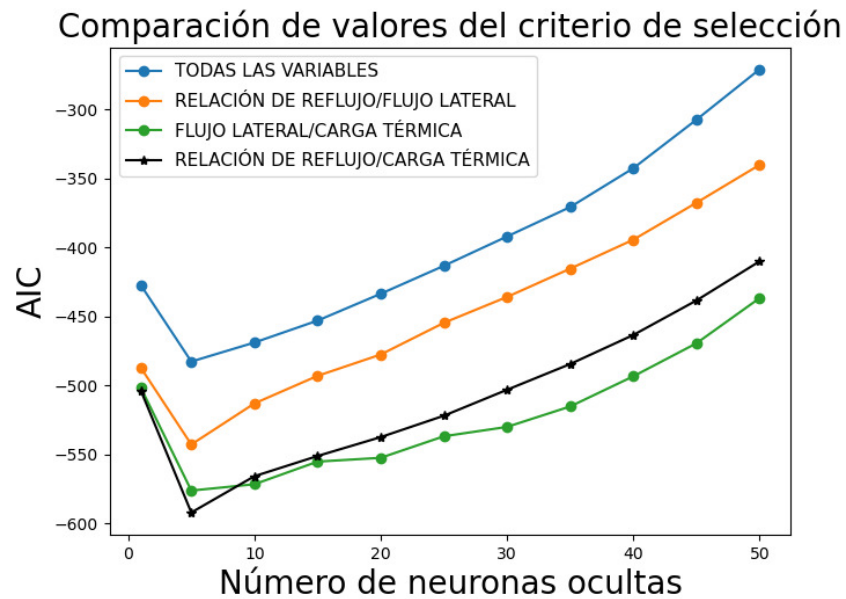


Figura 5.5 Valor obtenido de AIC con diferentes funciones de activación usando el optimizador RMSPROP para el sistema intensificado: a) Comparación, b) Función Tanh, c) Lineal y d) ReLU.

Se observa que la línea negra es el mejor modelo, teniendo como características que a la red neuronal solo se le alimentan las variables manipulables relación de reflujo y la carga térmica. Por lo que se prueba la capacidad de predicción prediciendo los perfiles de composición de la acetona, el etanol y el butanol en la Fig. 5.6.

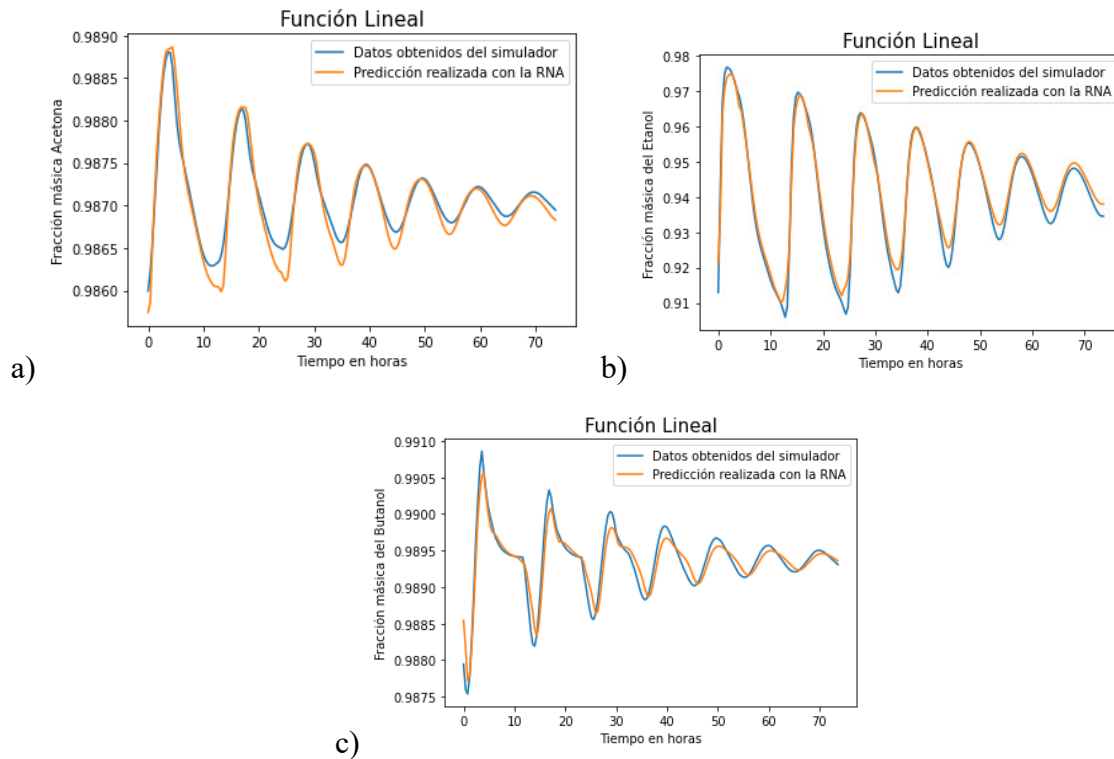


Figura 5.6 Predicción realizada alimentando la relación de reflujo y la carga térmica. En la Fig. 5.16 se puede observar cómo se obtienen buenas predicciones para los perfiles de los 3 componentes purificados en la columna de destilación con pared divisoria. La Fig. 5.7 se muestra las entradas y las salidas de la RNA resultante de la metodología hasta el momento, eliminando los valores del flujo lateral de la alimentación a la capa de entrada.

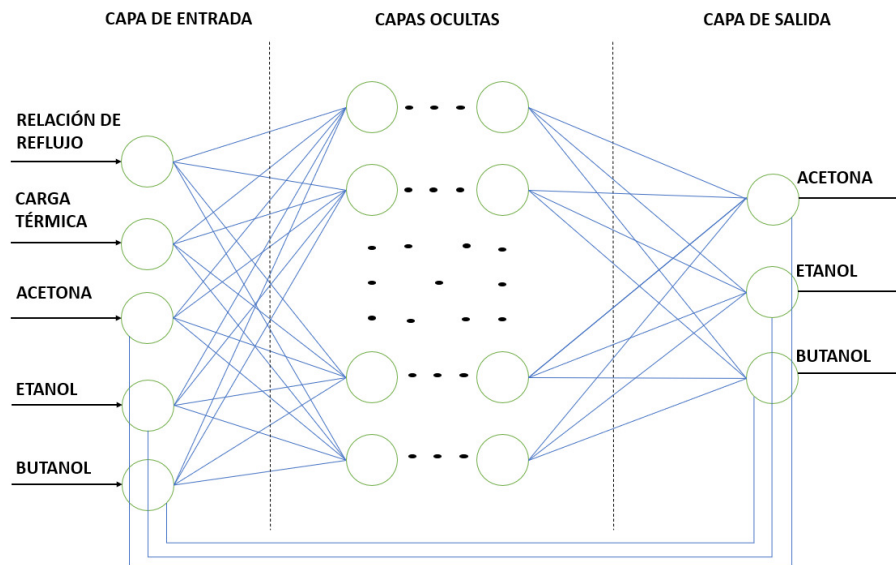


Figura 5.7 Diagrama de las variables con más peso alimentadas a la estructura de la RNA.

5.4 DETERMINACIÓN DEL NÚMERO DE CAPAS OCULTAS

En la Fig. 5.8 se compara el error obtenido al aumentar la complejidad de la RNA al añadir capas ocultas de neuronas. En esta sección se realiza el análisis a la red resultante de reducir las variables alimentadas a la capa de entrada, utilizando la mostrada en la Fig. 5.7. Mientras que en la Fig. 5.9 se muestra el valor del AIC para cada aumento de capas de la misma red.

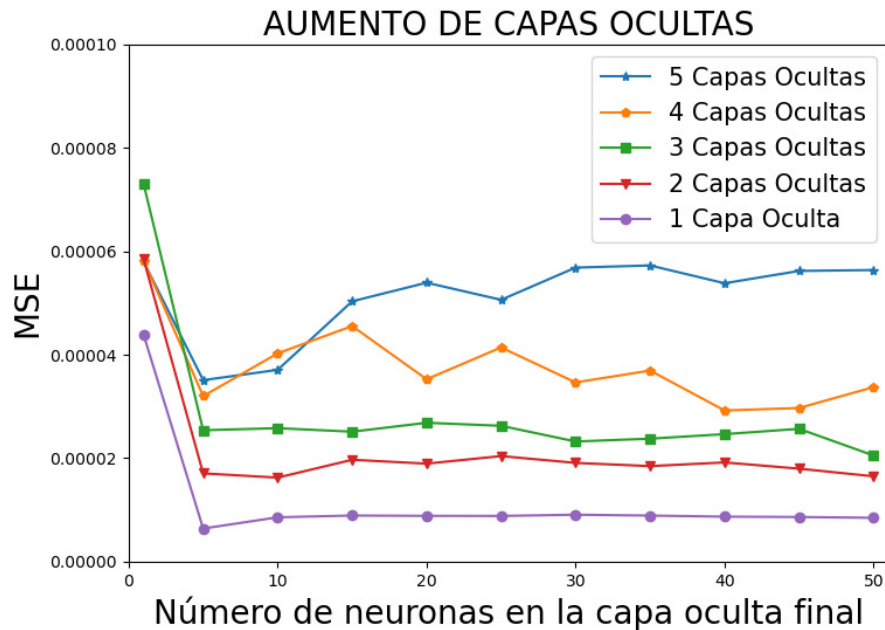


Figura 5.8 Comparación del valor del MSE al aumentar el número de capas ocultas a la RNA.

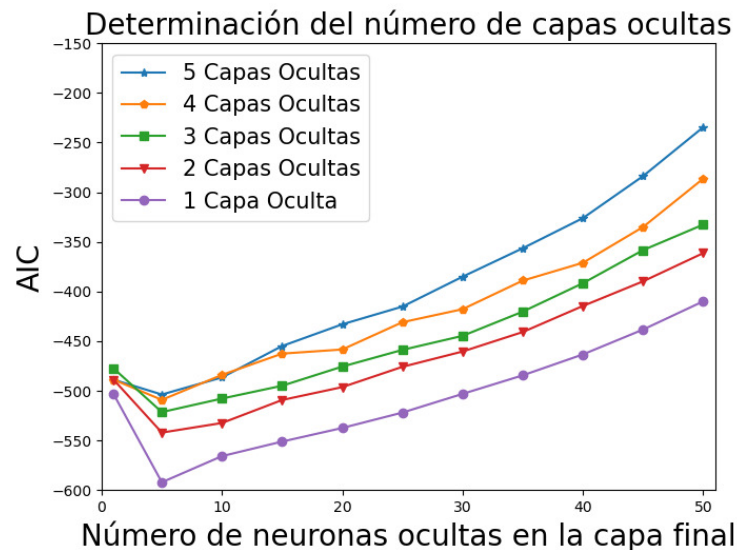


Figura 5.9 Comparación del valor del AIC al aumentar el número de capas ocultas.

5.5 ANÁLISIS DE RESULTADOS PARA EL SISTEMA INTENSIFICADO

Los resultados mostrados para la columna de pared divisoria del caso de estudio indican que una RNA con la función de activación lineal y el optimizador RMSPROP otorgan la mejor capacidad de predicción en comparación con las demás estructuras, así como que se reduce el valor del MSE al no alimentar los datos del flujo de salida lateral a la capa de entrada y que también se puede observar que se reduce el valor del AIC al disminuir la complejidad del modelo.

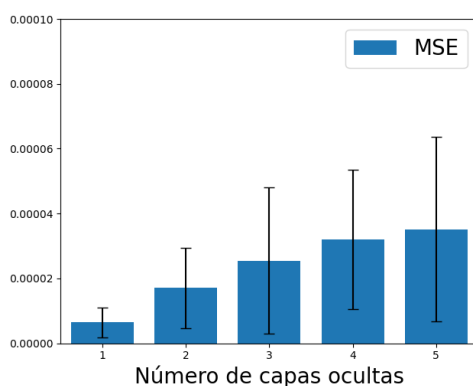


Figura 5.10 Evolución del valor del MSE al aumentar el número de capas ocultas con 5 neuronas en la capa final.

Por último, en la Fig. 5.10 es posible observar que en este estudio al aumentar el número de capas ocultas se aumenta el valor del error por lo que una capa es suficiente para obtener buenas predicciones, además de que así se obtiene el menor valor del AIC al tener 5 de neuronas en las capas ocultas, esto podría ser causado por un número pobre de épocas ya que se usó el mismo para todo número de capas ocultas. La Tabla 5.2 muestra la estructura de la RNA que mejores valores obtuvo a lo largo del estudio de la arquitectura de la red y sus predicciones pueden observarse en la Fig. 5.6.

Tabla 5.2. Resumen de la estructura de la RNA con mejor capacidad de modelado para el sistema intensificado.

Porcentaje de datos para entrenamiento	24%	Función de activación	Lineal
Porcentaje de datos para prueba	76%	Optimizador	RMSPROP
Tamaño de paso de retroalimentación	5	Variables manipulables alimentadas a la red	Relación de reflujo/ Carga Térmica
Número de variables alimentadas	5	Valor del MSE	1.396095×10^{-5}
Tipo de neuronas	LSTM	Valor del AIC	-591.96
Número de capas ocultas	1	Perfiles de composición Predichos	Acetona Etanol Butanol
Número de neuronas ocultas	5	Lenguaje de programación y librería	Python 3.8 Keras 2.12.0

CAPÍTULO 6

CONCLUSIONES

6.1 CONCLUSIONES

Se ha presentado un análisis de las respuestas del uso de una RNA para modelar la dinámica de una columna de destilación intensificada con una columna de pared divisoria para la obtención de biocombustibles purificados

Los resultados de la columna de pared divisoria indican que no es necesario alimentar las tres variables manipulables a la red neuronal artificial y que sólo es necesario alimentar dos, la relación de reflujo y la carga térmica, obteniendo un valor mínimo de 1.396095×10^{-5} del MSE, el cual es suficiente para modelar el sistema, y un valor de -591.96 de AIC con 5 neuronas en una única capa oculta, la función de activación lineal y el optimizador RMSPROP. Esto repercutiría en futuros trabajos al implementar sólo dos controladores para las tres variables de salida perturbadas del sistema intensificado, lo que reduciría considerablemente el coste de operación al tener menos controladores y con un bajo número de neuronas en la estructura de la red.

En general, es necesario analizar la estructura de una RNA al tratar de modelar y simular la dinámica de todos los procesos de separación usados para la obtención de biocombustibles, así como realizar el estudio en los procesos intensificados con el fin de reducir costos.

CAPÍTULO 7

RECOMENDACIONES

7.1 RECOMENDACIONES

Como trabajo futuro se propone el estudio de la dinámica de la columna de pared divisoria reduciendo a una el número de variables manipulables alimentadas a la capa de entrada manteniendo como respuesta los tres perfiles de composición a partir de la obtención de una serie de datos con perturbaciones de alto impacto para poder modelar el sistema de una manera más precisa. Existen una gran variedad de funciones de activación y optimizadores en la literatura que se podrían analizar para poder optimizar la red neuronal así como distintos tipos de neuronas.

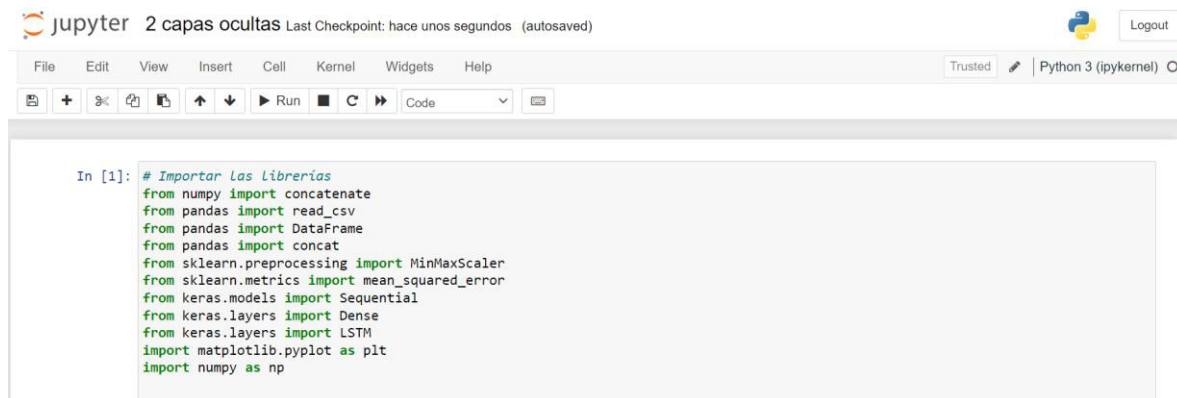
CAPÍTULO 8
APÉNDICE A
CÓDIGOS DE PROGRAMACIÓN

8.1 APÉNDICE A

Como se explicó anteriormente se hace uso del lenguaje de programación PYTHON 3.8 en su entorno ANACONDA. En caso de que su equipo de cómputo tenga una GPU puede activarla y usarla para poder ejecutar los códigos de la librería de TENSORFLOW creando otro entorno de distribución en ANACONDA NAVIGATOR. Aunque no se recomienda ya que en caso de que el código ejecutado sea complejo puede producir un calentamiento.

En este estudio se utilizó JUPYTER NOTEBOOK el cual es un entorno de programación amigable al poder mostrar una serie de gráficas en un mismo archivo y mostrar los errores cometidos al momento de codificar.

En la primera línea se llaman las librerías usadas en el estudio realizado. El símbolo “#” hace referencia a que se realiza un comentario que no se toma en cuenta al ejecutar el código y se recomienda comentar todas las líneas del programa para no olvidar que función realizan.



```
In [1]: # Importar Las Librerías
from numpy import concatenate
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import matplotlib.pyplot as plt
import numpy as np
```

De algunas librerías se importan ciertos paquetes que se usarán a lo largo del código, mientras que cuando se inicia con “import” es porque se importara la librería completa llamándola de alguno nombre como “plt” o “np”. Las librerías de NUMPY y PANDAS se usaron para la manipulación de datos mientras que SKLEARN y KERAS fueron usadas para cálculos estadísticos y para desarrollar la red neuronal. La librería MATPLOTLIB se usó para crear gráficos que explicaran de forma detallada los resultados obtenidos.

A continuación, se muestra la línea de código donde se define la función “Iteration” para cada neurona donde se desarrolle el código. Como se muestra se calcula el valor del MSE 100 veces y se guarda en una lista denominada “List_MSE”.

Del archivo con terminación “.csv” se llama la información obtenida del simulador ASPEN Dynamics imprimiéndolo en un dataset que es posible observar y manipular en el código a gusto del usuario, como para poder obtener la Fig. 2.6. Una vez obtenidos los valores en una variable denominada “values” se convierten las series a un aprendizaje supervisado y con una línea del código se aseguran de que los datos son del tipo float.

Con la función MINMAXSCALER importada de la librería SKLEARN se realiza la normalización de datos con valores en un rango del 0 a 1.

Dentro de las características de las redes tipo LSTM se habla de un tiempo de retraso, en este caso en la variable “n_lag” se indica que el valor de esa característica es de 5, y después se indica en la variable “n_features” que el número de variables alimentadas es de 5. Antes de realizar el acomodo matricial se indica que el número de datos para cada variable alimentada es de 61, ya que en este caso se tienen 24 horas de entrenamiento y 76 de prueba para los datos con un paso de 0.4 horas.

Ya en el diseño de la red se indica con la variable model que se trata de uno Secuencial, donde es posible agregar las capas de neuronas indicando su tipo, en este caso LSTM junto con el número de neuronas, la función de activación y ciertas indicaciones para que las operaciones matriciales no dieran errores al momento de entrenar la red. Al agregar la siguiente capa de neuronas de igual manera se indica el tipo y el número de neuronas se da por la variable Neurons previamente indicada al inicio de la función. La capa densa indica que tiene 3 salidas de la red para poder obtener la respuesta de la Acetona, el Etanol y el Butanol. Finalmente se indica la función de pérdida y el optimizador de la red.

Por último, se entrena la red con un número definido de épocas que en este caso es 3000 y se manipulan los resultados para obtener ser agregados a la lista “List_MSE” la cual se podrá manipular fácilmente para obtener gráficos.

```
def Iteration(Neurons):
    List_MSE=[]
    for i in range(1,101):
        dataset = read_csv('Datos_Columna.csv', header=0, index_col=0,nrows=251)
        dataset = dataset[["Acetona %w","Etanol %w","Butanol %w","Reflujo","Carga
térmica"]]
        values = dataset.values
        # convert series to supervised learning
        def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
            n_vars = 1 if type(data) is list else data.shape[1]
            df = DataFrame(data)
            cols, names = list(), list()
            # input sequence (t-n, ... t-1)
            for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
                names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
            # forecast sequence (t, t+1, ... t+n)
            for i in range(0, n_out):
                cols.append(df.shift(-i))
                if i == 0:
                    names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
                else:
                    names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
            # put it all together
            agg = concat(cols, axis=1)
```

```

agg.columns = names
# drop rows with NaN values
if dropnan:
    agg.dropna(inplace=True)
return agg

# asegurarse de que todos los datos son tipo float
values = values.astype('float32')
# normalizar datos
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# especificar el número de horas de retraso
n_lag = 5
n_features = 5
# frame as supervised learning
reframed = series_to_supervised(scaled, n_lag, 1)
# Dividir en entrenamiento y prueba
values = reframed.values
n_train_index = 61
train = values[:n_train_index, :]
test = values[n_train_index:, :]
# Dividir en entradas y salidas
#CHANGES HERE
#split into input and outputs
n_obs = n_lag * n_features
train_X = train[:, :n_obs]
train_y = train[:, -n_features:(-n_features+3)]
test_X = test[:, :n_obs]
test_y = test[:, -n_features:(-n_features+3)]

train_X = train_X.reshape((train_X.shape[0], n_lag, n_features))
test_X = test_X.reshape((test_X.shape[0], n_lag, n_features))

#CAMBIOS AQUI

#design network
model = Sequential()
model.add(LSTM(5,activation='linear', input_shape=(train_X.shape[1],
train_X.shape[2]), return_sequences=True))
model.add(LSTM(Neurons, activation='linear'))
model.add(Dense(3))
model.compile(loss='mse', optimizer='rmsprop')

#fit network
history = model.fit(train_X, train_y, epochs=3000, validation_data=(test_X, test_y),
verbose=0, shuffle=False)

```

```

#make a prediction
y_hat = model.predict(test_X)

#CHANGES HERE
test_X = test_X.reshape((test_X.shape[0], n_lag*n_features))
inv_yhat = concatenate((y_hat, test_X[:, -2:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0:3]

#CHANGES HERE
#invert scaling for actual
test_y = test_y.reshape((len(test_y), 3))
inv_y = concatenate((test_y, test_X[:, -2:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0:3]
#CHANGES HERE
#calculate MSE

mse = mean_squared_error(inv_y[:, (0,1,2)], inv_yhat[:, (0,1,2)])
List_MSE.append(mse)
return List_MSE

```

Se realiza el análisis ejecutando el siguiente código:

```

In [43]: Lista_MSE_1 = Iteration(1)
PROM_MSE_1 = sum(Lista_MSE_1)/len(Lista_MSE_1)
DESV_MSE_1 = np.std(Lista_MSE_1)
print('Lista_MSE_1')
print(Lista_MSE_1)
print('PROM_MSE_1')
print(PROM_MSE_1)
print('DESV_MSE_1')
print(DESV_MSE_1)

```

En el cual se tiene una neurona en la segunda capa oculta para la red descrita en la función Iteración. Ya que el proceso se repetirá 100 veces se calcula el valor promedio del cálculo del Error Cuadrático Promedio (MSE) y su desviación estándar.

Este procedimiento se realiza aumentando el número de neuronas de 1,5,10,15, etc. Hasta llegar un valor de 50 neuronas.

```
In [43]: Lista_MSE_1 = Iteration(1)
PROM_MSE_1 = sum(Lista_MSE_1)/len(Lista_MSE_1)
DESV_MSE_1 = np.std(Lista_MSE_1)
print('Lista_MSE_1')
print(Lista_MSE_1)
print('PROM_MSE_1')
print(PROM_MSE_1)
print('DESV_MSE_1')
print(DESV_MSE_1)
```

```
Lista_MSE_1
[7.681279e-05, 4.24345e-05, 4.4409553e-05, 8.4872096e-05, 8.104999e-05, 5.1858882e-05, 3.8844995e-05, 6.782123e-05, 2.7091495e-05, 5.921278e-05, 5.8182162e-05, 5.432266e-05, 3.0885585e-05, 6.195765e-05, 8.07295e-05, 5.810278e-05, 6.720524e-05, 4.794562e-05, 6.974502e-05, 6.517787e-05, 4.3571392e-05, 7.579516e-05, 6.485749e-05, 5.4995424e-05, 5.3656968e-05, 6.4571876e-05, 7.439653e-05, 8.4006395e-05, 8.22749e-05, 4.2804222e-05, 8.28109e-05, 7.432967e-05, 7.9450016e-05, 6.4701475e-05, 5.4667395e-05, 5.340864e-05, 5.9171605e-05, 8.0109305e-05, 3.9926173e-05, 7.579668e-05, 4.9912123e-05, 6.0824765e-05, 6.540935e-05, 4.207948e-05, 3.6071644e-05, 4.9059225e-05, 3.292454e-05, 6.9218564e-05, 4.5145935e-05, 3.3312735e-05, 5.269133e-05, 2.5923046e-05, 2.9183722e-05, 6.188977e-05, 7.55248e-05, 7.299879e-05, 7.3888375e-05, 6.862988e-05, 9.622433e-05, 4.1263353e-05, 7.2587405e-05, 7.173706e-05, 5.374776e-05, 3.2961936e-05, 5.9877824e-05, 9.162886e-05, 3.8896087e-05, 6.540381e-05, 5.9404712e-05, 4.687118e-05, 4.2289914e-05, 2.5107996e-05, 5.618439e-05, 3.2467193e-05, 2.6845286e-05, 3.592668e-05, 7.4135976e-05, 7.423528e-05, 6.929649e-05, 7.5308424e-05, 3.6414265e-05, 5.3784446e-05, 4.62509e-05, 8.8475404e-05, 2.4544357e-05, 4.9379683e-05, 6.715319e-05, 6.6193046e-05, 3.3872468e-05, 3.2410084e-05, 6.413755e-05, 7.07578e-05, 3.522106e-05, 6.758023e-05, 6.483242e-05, 9.279826e-05, 9.2052294e-05, 5.8593254e-05, 8.443837e-05, 5.9013186e-05]
PROM_MSE_1
5.848956900081248e-05
DESV_MSE_1
1.7966402e-05
```

```
In [44]: Lista_MSE_5 = Iteration(5)
PROM_MSE_5 = sum(Lista_MSE_5)/len(Lista_MSE_5)
DESV_MSE_5 = np.std(Lista_MSE_5)
print('Lista_MSE_5')
print(Lista_MSE_5)
print('PROM_MSE_5')
print(PROM_MSE_5)
print('DESV_MSE_5')
print(DESV_MSE_5)
```

```
Lista_MSE_5
[1.9063926e-05, 1.529461e-05, 6.05101e-06, 5.4810066e-06, 2.7557175e-05, 1.6978584e-05, 1.0032775e-05, 6.128001e-05, 3.319319e-05, 1.569503e-05, 3.3859596e-05, 7.184757e-06, 2.0339703e-05, 1.0957148e-05, 5.942559e-06, 1.9250796e-05, 1.2083431e-05, 1.0013945e-05, 1.4811872e-05, 2.1719565e-05, 1.1908492e-05, 3.8006347e-06, 1.3965045e-05, 1.0985943e-05, 3.5851237e-05, 1.07591295e-05, 1.6789769e-05, 1.3397548e-05, 1.6570219e-05, 2.0393987e-05, 1.1011359e-05, 1.1869208e-05, 3.1024156e-05, 2.3172703e-05, 7.5341054e-06, 7.563676e-06, 3.707212e-05, 1.2051577e-05, 5.7485286e-06, 8.799e-06, 4.0577197e-05, 8.818289e-06, 3.349927e-05, 1.7278515e-05, 3.8899416e-06, 1.22228885e-05, 4.0722924e-05, 2.1465623e-05, 4.226253e-06, 3.014099e-05, 1.0236095e-05, 4.2819347e-06, 2.6149879e-05, 3.6564636e-06, 1.7180128e-05, 7.0015917e-06, 7.713894e-06, 4.1546333e-05, 1.9886049e-05, 2.7060958e-05, 1.3185577e-05, 1.0873203e-05, 3.7061876e-05, 1.4383196e-05, 1.0515396e-05, 1.1748453e-05, 1.2418391e-05, 1.3773041e-05, 2.8175024e-05, 1.4907936e-05, 1.4103288e-05, 1.5902999e-05, 1.9177685e-05, 8.368173e-06, 2.2920032e-05, 1.991605e-05, 2.5275482e-05, 8.719492e-06, 6.8851136e-06, 8.019356e-06, 9.567105e-06, 5.881055e-06, 6.5231693e-06, 7.4975856e-06, 1.3323291e-05, 1.0942583e-05, 9.155429e-06, 4.5721074e-05, 9.653005e-06, 1.688292e-05, 1.0674295e-05, 1.1394243e-05, 8.03766e-06, 2.8552371e-05, 1.3771111e-05, 7.0669626e-06, 2.3000752e-05, 3.4836928e-06, 7.7618104e-05, 1.6664215e-05]
PROM_MSE_5
1.706354720681702e-05
DESV_MSE_5
1.2344372e-05
```

```
In [45]: Lista_MSE_10 = Iteration(10)
PROM_MSE_10 = sum(Lista_MSE_10)/len(Lista_MSE_10)
DESV_MSE_10 = np.std(Lista_MSE_10)
print('Lista_MSE_10')
print(Lista_MSE_10)
print('PROM_MSE_10')
print(PROM_MSE_10)
print('DESV_MSE_10')
print(DESV_MSE_10)
```

```
Lista_MSE_10
[1.3026663e-05, 5.723155e-05, 1.1459223e-05, 1.2557925e-05, 8.239634e-06, 5.4314864e-05, 1.1821285e-05, 1.9325187e-05, 1.9136203e-05, 5.3614126e-06, 1.5435942e-05, 5.1050806e-06, 6.427794e-06, 1.2350375e-05, 6.402259e-06, 1.6010908e-05, 2.9748975e-05, 1.7608681e-05, 1.1913567e-05, 2.7543292e-05, 2.3179628e-05, 2.0722664e-05, 1.6717933e-05, 6.646014e-06, 8.823618e-06, 2.4298737e-05, 1.8660328e-05, 7.470693e-06, 2.4130824e-05, 1.9974372e-05, 4.3509353e-05, 1.0856456e-05, 5.088077e-06, 1.0202953e-05, 1.6515603e-05, 3.8626316e-05, 2.3338414e-05, 9.399523e-06, 1.1987525e-05, 2.4690957e-05, 1.1037621e-05, 4.8625648e-06, 2.8728651e-05, 5.6152094e-06, 1.24450935e-05, 2.0746047e-05, 2.0172713e-05, 3.2744847e-06, 1.591745e-05, 1.8394501e-05, 6.0037496e-06, 4.151634e-05, 8.423428e-06, 3.4974266e-05, 1.4877715e-05, 4.894508e-06, 4.599398e-06, 6.967954e-06, 2.7828657e-06, 9.372165e-06, 8.08496e-06, 9.3695335e-06, 9.464257e-06, 1.8275137e-05, 1.5246314e-05, 8.5051615e-06, 3.930579e-05, 1.3054475e-05, 1.1108664e-05, 3.093349e-05, 2.2211769e-05, 1.0395236e-05, 1.5747588e-05, 1.0690321e-05, 7.7236555e-06, 9.691777e-06, 6.7928454e-06, 1.9053312e-05, 1.7619457e-05, 1.1598292e-05, 1.6600019e-05, 1.4609911e-05, 1.2942422e-05, 9.653435e-06, 1.3679112e-05, 7.361074e-06, 3.336065e-05, 1.3652921e-05, 1.0967557e-05, 1.1575797e-05, 7.946698e-06, 1.669891e-05, 4.9022405e-05, 1.4004097e-05, 1.0044007e-05, 2.0691261e-05, 7.1787804e-06, 6.1672486e-06, 3.906192e-05, 1.22495485e-05]
PROM_MSE_10
1.624330882123104e-05
DESV_MSE_10
1.1073417e-05
```

```
In [46]: Lista_MSE_15 = Iteration(15)
PROM_MSE_15 = sum(Lista_MSE_15)/len(Lista_MSE_15)
DESV_MSE_15 = np.std(Lista_MSE_15)
print('Lista_MSE_15')
print(Lista_MSE_15)
print('PROM_MSE_15')
print(PROM_MSE_15)
print('DESV_MSE_15')
print(DESV_MSE_15)
```

Lista_MSE_15
[1.8574634e-05, 9.377712e-06, 5.3204385e-06, 2.5023466e-05, 1.4288043e-05, 1.0148316e-05, 1.9612306e-05, 3.080115e-05, 1.7072078e-05, 3.7588903e-05, 1.7684752e-05, 3.1078875e-05, 6.870096e-06, 7.090332e-06, 6.6604116e-06, 3.28729e-05, 7.4647355e-06, 1.133404e-05, 8.07983e-06, 3.10657e-05, 1.4452348e-05, 4.316945e-05, 2.4087502e-05, 2.4085397e-05, 6.219951e-06, 3.0215231e-05, 6.3976127e-06, 1.398483e-05, 2.2617576e-05, 9.48938e-06, 6.886628e-06, 1.4338664e-05, 6.4829915e-06, 2.71076e-05, 1.41469545e-05, 1.05817935e-05, 1.1698837e-05, 3.522658e-05, 2.529079e-05, 3.510669e-05, 2.045925e-05, 1.35769615e-05, 1.4170625e-05, 5.7575576e-06, 2.514045e-05, 9.7917045e-06, 2.6358599e-05, 5.4477205e-05, 2.3759458e-05, 2.2281924e-05, 9.58152e-06, 3.210288e-05, 3.423869e-06, 1.7412804e-05, 7.273417e-06, 4.5645545e-05, 2.4801098e-05, 1.4094462e-05, 3.8074304e-05, 2.2204205e-05, 2.9671375e-05, 2.9759745e-05, 1.854797e-05, 1.7485609e-05, 1.2225851e-05, 3.5880665e-05, 7.340061e-06, 8.6420905e-06, 4.5297944e-05, 8.38700e-06, 6.0164675e-06, 4.697593e-05, 5.344296e-06, 8.720729e-06, 9.409373e-06, 6.7700876e-06, 1.5512132e-05, 4.5483775e-05, 3.334568e-05, 1.906871e-05, 2.4219305e-05, 2.3402368e-05, 3.86027e-05, 2.3168106e-05, 1.192922e-05, 2.5944688e-05, 1.9273144e-05, 9.071999e-06, 4.1879607e-06, 6.350057e-06, 2.1522683e-05, 4.577285e-05, 2.8801436e-05, 7.1400978e-06, 8.281595e-06, 8.101753e-06, 3.4149285e-05, 5.2973223e-06, 1.4408163e-05, 9.413585e-06]
PROM_MSE_15
1.9697220036505314e-05
DESV_MSE_15
1.21101375e-05

```
In [47]: Lista_MSE_20 = Iteration(20)
PROM_MSE_20 = sum(Lista_MSE_20)/len(Lista_MSE_20)
DESV_MSE_20 = np.std(Lista_MSE_20)
print('Lista_MSE_20')
print(Lista_MSE_20)
print('PROM_MSE_20')
print(PROM_MSE_20)
print('DESV_MSE_20')
print(DESV_MSE_20)
```

Lista_MSE_20
[3.3266777e-05, 2.7246306e-05, 2.9298617e-05, 3.2086357e-06, 2.3077419e-05, 1.4319289e-05, 1.8614135e-05, 2.1651489e-05, 2.0093303e-05, 1.4077435e-05, 2.3010134e-05, 1.004428e-05, 3.922026e-05, 1.2582629e-05, 4.2268425e-06, 4.32457e-06, 6.9755624e-06, 1.1422851e-05, 6.578882e-06, 1.1672128e-05, 4.000379e-05, 2.0616675e-05, 3.668723e-05, 1.440944e-05, 1.0921079e-05, 5.3510225e-06, 2.7534166e-05, 1.15397115e-05, 2.2386228e-05, 2.4744315e-05, 1.5548234e-05, 8.10499e-06, 8.391192e-06, 1.2137026e-05, 1.2638361e-05, 2.2731117e-05, 9.086333e-06, 2.5222653e-05, 1.742099e-05, 2.5102034e-05, 1.9858644e-05, 8.402322e-06, 1.27923195e-05, 8.996689e-06, 1.6895216e-05, 1.3660872e-05, 2.8949047e-05, 1.38769465e-05, 1.1820607e-05, 2.5174935e-05, 3.343844e-05, 5.428368e-06, 5.266121e-06, 2.6949985e-05, 6.9270354e-06, 4.7012113e-06, 2.3167673e-05, 2.33394e-05, 1.8030085e-05, 1.4273634e-05, 1.3931287e-05, 9.319626e-06, 1.2190761e-05, 1.6293792e-05, 5.380825e-06, 1.169834e-05, 1.982078e-05, 8.879979e-06, 3.896803e-05, 8.462401e-05, 9.350199e-06, 2.2526794e-05, 2.0747679e-05, 5.7873985e-06, 1.4243062e-05, 2.1633692e-05, 1.907924e-05, 7.772426e-06, 3.6855577e-05, 1.2969996e-05, 4.853957e-06, 2.535834e-05, 2.2201777e-05, 3.5254525e-05, 6.7263713e-06, 1.7173357e-05, 4.696212e-05, 3.161845e-05, 2.0807916e-05, 1.8304532e-05, 1.8444875e-05, 8.013031e-06, 6.279656e-05, 1.1212371e-05, 1.5392125e-05, 2.0796748e-05, 2.5035699e-05, 2.9811723e-05, 8.840445e-06, 4.4707074e-05]
PROM_MSE_20
1.8947345108699664e-05
DESV_MSE_20
1.2654298e-05

```
In [48]: Lista_MSE_25 = Iteration(25)
PROM_MSE_25 = sum(Lista_MSE_25)/len(Lista_MSE_25)
DESV_MSE_25 = np.std(Lista_MSE_25)
print('Lista_MSE_25')
print(Lista_MSE_25)
print('PROM_MSE_25')
print(PROM_MSE_25)
print('DESV_MSE_25')
print(DESV_MSE_25)
```

Lista_MSE_25
[1.5467616e-05, 1.0468814e-05, 1.7537564e-05, 2.2711894e-05, 1.7480066e-05, 2.2822314e-05, 1.4070202e-05, 1.882854e-05, 1.5541114e-05, 3.3940043e-05, 1.5722233e-05, 5.7987454e-06, 2.4732888e-05, 3.1886637e-05, 1.8522782e-05, 9.120073e-06, 1.26235755e-05, 2.327239e-05, 1.9950445e-05, 4.6327355e-06, 2.8247678e-05, 2.0924177e-05, 6.144169e-05, 1.5810869e-05, 1.7166609e-05, 1.4301935e-05, 3.3236905e-05, 7.966651e-06, 2.5314248e-05, 2.5313062e-05, 1.7694001e-05, 2.4866284e-05, 2.0976384e-05, 1.4062781e-05, 9.0103e-06, 2.0284851e-05, 1.6619531e-05, 3.0280004e-05, 3.4176184e-05, 1.4248143e-05, 3.6442525e-05, 5.1483917e-06, 2.2441147e-05, 6.7305555e-05, 3.525099e-05, 2.0937392e-05, 1.04215205e-05, 3.4233864e-05, 1.8992214e-05, 1.5899717e-05, 3.2992426e-05, 1.945807e-05, 1.8944207e-05, 3.8734365e-06, 2.1478632e-05, 1.1766534e-05, 1.7890432e-05, 1.5953512e-05, 1.5662268e-05, 2.1154236e-05, 3.9238173e-05, 1.5311278e-05, 3.3618457e-05, 1.4530965e-05, 7.132059e-06, 2.1115831e-05, 1.1880768e-05, 1.3643999e-05, 2.0062778e-05, 1.8269558e-05, 2.8008117e-05, 1.9520225e-05, 1.0854488e-05, 9.153485e-06, 1.957552e-05, 2.0520898e-05, 1.4730359e-05, 1.2313791e-05, 1.7536935e-05, 1.3017766e-05, 1.913908e-05, 4.1393476e-05, 1.8227021e-05, 2.712958e-05, 1.2827518e-05, 9.249498e-06, 6.349374e-06, 2.9047062e-05, 1.9489693e-05, 5.5786513e-06, 1.5395055e-05, 2.7940323e-05, 7.734808e-06, 5.7975518e-05, 3.116019e-05, 1.6219628e-05, 1.389271e-05, 1.3392141e-05, 2.0228419e-05, 3.5146295e-05]
PROM_MSE_25
2.040871524513932e-05
DESV_MSE_25
1.1104764e-05


```
In [49]: Lista_MSE_30 = Iteration(30)
PROM_MSE_30 = sum(Lista_MSE_30)/len(Lista_MSE_30)
DESV_MSE_30 = np.std(Lista_MSE_30)
print('Lista_MSE_30')
print(Lista_MSE_30)
print('PROM_MSE_30')
print(PROM_MSE_30)
print('DESV_MSE_30')
print(DESV_MSE_30)
```

```
Lista_MSE_30
[8.339915e-06, 3.4416993e-05, 2.0861371e-05, 1.45920085e-05, 2.2894694e-05, 6.963422e-06, 1.7841798e-05, 1.4946517e-05, 1.21671
22e-05, 9.690629e-06, 2.5352378e-05, 1.14032955e-05, 1.2190717e-05, 1.2649438e-05, 1.27317335e-05, 6.6820467e-06, 2.6472038e-0
5, 1.6209777e-05, 4.8781108e-06, 1.8514329e-05, 9.748679e-06, 1.9600924e-05, 2.5921605e-05, 6.6693106e-06, 1.0824396e-05, 3.014
7072e-05, 2.192318e-05, 1.4035449e-05, 1.02249205e-05, 8.914188e-06, 6.6971515e-06, 4.075242e-05, 1.8562294e-05, 2.6841999e-05,
1.3557932e-05, 6.6608045e-06, 1.106007e-05, 8.809845e-06, 3.2655942e-05, 9.761082e-06, 9.487671e-06, 2.5846623e-05, 3.7800335e-
05, 2.3920424e-05, 4.1305284e-06, 1.9810313e-05, 2.2534778e-05, 9.98968e-06, 1.5348469e-05, 8.113132e-06, 1.5526999e-05, 2.2315
608e-05, 1.2630951e-05, 9.777824e-06, 7.5462635e-06, 1.8662378e-05, 2.0822712e-05, 1.6913433e-05, 3.7418184e-05, 1.3474743e-05,
3.614433e-05, 1.1518681e-05, 1.3234197e-05, 1.75152e-05, 1.1803661e-05, 3.592068e-05, 1.7882743e-05, 6.8631584e-06, 8.627233e-0
6, 1.1873203e-05, 4.8701568e-06, 2.2382226e-05, 2.0721762e-05, 1.020174e-05, 8.3192695e-05, 5.2039642e-05, 1.3345999e-05, 2.759
8944e-05, 6.260369e-06, 1.943845e-05, 4.3908512e-05, 4.038278e-05, 1.1866222e-05, 8.196478e-06, 2.1566653e-05, 1.2022687e-05,
7.047301e-05, 1.1307722e-05, 1.975586e-05, 2.8355344e-06, 1.5152948e-05, 2.82715e-05, 2.9500268e-05, 1.108784e-05, 1.3078564e-0
5, 1.3327953e-05, 4.8718994e-05, 5.1365623e-06, 2.9107856e-05, 4.674812e-05]
PROM_MSE_30
1.909119779156754e-05
DESV_MSE_30
1.3528145e-05
```

```
In [50]: Lista_MSE_35 = Iteration(35)
PROM_MSE_35 = sum(Lista_MSE_35)/len(Lista_MSE_35)
DESV_MSE_35 = np.std(Lista_MSE_35)
print('Lista_MSE_35')
print(Lista_MSE_35)
print('PROM_MSE_35')
print(PROM_MSE_35)
print('DESV_MSE_35')
print(DESV_MSE_35)
```

```
Lista_MSE_35
[2.0087205e-05, 9.24687e-06, 2.1627302e-05, 6.581653e-06, 1.0364936e-05, 1.9266941e-05, 7.922613e-06, 5.0489933e-05, 1.9291389e-
05, 1.0801093e-05, 3.841985e-05, 2.5318526e-05, 1.01480555e-05, 3.608785e-05, 7.349266e-05, 1.2590772e-05, 3.1345906e-05, 1.37
07165e-05, 3.2019194e-05, 7.528182e-06, 2.3233746e-05, 1.4603814e-05, 1.5813821e-05, 2.1257909e-05, 8.094312e-06, 1.6794194e-0
5, 2.043204e-05, 2.0379526e-05, 1.4728178e-05, 7.1583604e-06, 4.3372336e-05, 3.4442884e-05, 1.6397711e-05, 3.295978e-05, 4.345
5733e-05, 1.2578338e-05, 8.401986e-06, 2.7823897e-05, 1.02036765e-05, 2.181479e-05, 4.8299746e-05, 2.1958824e-05, 1.60412e-05,
2.5517358e-05, 2.2542694e-05, 8.079572e-06, 2.6427188e-05, 5.399033e-06, 9.386476e-06, 5.3154654e-05, 4.918456e-06, 3.189444e-0
5, 2.3044357e-05, 1.5843741e-05, 7.748517e-06, 1.2006451e-05, 9.766404e-06, 2.4099436e-05, 1.0960939e-05, 1.8539691e-05, 1.1955
2315e-05, 4.4488283e-06, 4.9957766e-06, 1.7644625e-05, 8.5075735e-06, 6.9025186e-06, 3.4481298e-05, 1.2689618e-05, 8.121186e-0
6, 7.647138e-06, 1.0154886e-05, 1.4219341e-05, 2.3918872e-05, 1.4915288e-05, 4.544566e-06, 2.0380272e-05, 3.5910063e-05, 2.1199
283e-05, 1.5015109e-05, 1.0933767e-05, 2.0822617e-05, 1.4689544e-05, 1.1002128e-05, 2.2279803e-05, 2.6940215e-05, 9.002144e-06,
1.8010027e-05, 1.3374029e-05, 1.035117e-05, 1.6518736e-05, 8.249851e-06, 9.226314e-06, 1.4861863e-05, 4.6422447e-06, 1.4354544e-
05, 8.502685e-06, 1.2710169e-05, 1.3771386e-05, 2.0599584e-05, 4.2131607e-05]
PROM_MSE_35
1.8468749581188603e-05
DESV_MSE_35
1.2121503e-05
```

```
In [51]: Lista_MSE_40 = Iteration(40)
PROM_MSE_40 = sum(Lista_MSE_40)/len(Lista_MSE_40)
DESV_MSE_40 = np.std(Lista_MSE_40)
print('Lista_MSE_40')
print(Lista_MSE_40)
print('PROM_MSE_40')
print(PROM_MSE_40)
print('DESV_MSE_40')
print(DESV_MSE_40)
```

```
Lista_MSE_40
[1.20992145e-05, 2.4271663e-05, 9.346529e-06, 1.6423528e-05, 1.1954654e-05, 9.755532e-06, 2.5399213e-05, 2.583274e-05, 8.0488e-
06, 1.3538131e-05, 1.111383e-05, 2.9265777e-05, 1.741464e-05, 1.203606e-05, 2.3235183e-05, 1.6842007e-05, 1.3801294e-05, 1.3313
2535e-05, 6.9352277e-06, 1.8597038e-05, 2.551922e-05, 1.5604337e-05, 2.1532238e-05, 1.0668853e-05, 1.3296347e-05, 3.1616455e-0
5, 1.4572913e-05, 2.7559096e-05, 1.6805832e-05, 2.5012918e-05, 1.1765304e-05, 2.3199138e-05, 3.2091873e-05, 1.9288495e-05, 2.15
69342e-05, 1.3035659e-05, 1.2531141e-05, 2.3210878e-05, 4.0774856e-05, 3.3789634e-05, 2.411005e-05, 1.9070505e-05, 2.0533756e-0
5, 3.198824e-05, 4.9728355e-06, 3.9678845e-05, 1.625372e-05, 2.2306833e-05, 2.5250001e-05, 1.9160087e-05, 1.533837e-05, 2.11780
19e-05, 1.8395272e-05, 2.0949577e-05, 1.0307785e-05, 1.2598691e-05, 4.211826e-06, 2.4162307e-05, 2.3093315e-05, 2.8486385e-05,
1.8928993e-05, 1.7838958e-05, 1.1490112e-05, 1.4546019e-05, 6.893073e-06, 1.8544944e-05, 1.2013493e-05, 1.9197729e-05, 1.386228
6e-05, 8.077191e-06, 1.84107e-05, 7.917956e-06, 8.151336e-06, 1.16869205e-05, 3.337094e-05, 2.5946634e-05, 1.2856045e-05, 1.960
1872e-05, 3.4404664e-05, 7.0511837e-06, 1.6822052e-05, 2.0912843e-05, 3.9795665e-05, 1.083095e-05, 1.5159355e-05, 3.821305e-05,
2.372528e-05, 2.3172763e-05, 2.3391729e-05, 1.2178523e-05, 6.2876406e-06, 2.3892273e-05, 3.0944146e-05, 2.6845986e-05, 8.221224
e-06, 3.522386e-05, 1.8407973e-05, 3.9663868e-05, 1.3428602e-05, 1.51711965e-05]
PROM_MSE_40
1.9180938143108505e-05
DESV_MSE_40
8.65303e-06
```



```
In [52]: Lista_MSE_45 = Iteration(45)
PROM_MSE_45 = sum(Lista_MSE_45)/len(Lista_MSE_45)
DESV_MSE_45 = np.std(Lista_MSE_45)
print('Lista_MSE_45')
print(Lista_MSE_45)
print('PROM_MSE_45')
print(PROM_MSE_45)
print('DESV_MSE_45')
print(DESV_MSE_45)
```

```
Lista_MSE_45
[1.3042649e-05, 1.2466909e-05, 9.190838e-06, 1.2929005e-05, 2.0453926e-05, 3.0389952e-05, 1.6698848e-05, 1.957838e-05, 2.677348
5e-05, 1.7752687e-05, 2.6607318e-05, 1.3072736e-05, 3.7140802e-05, 9.892092e-06, 1.2146946e-05, 3.039678e-05, 1.843811e-05, 9.2
65261e-06, 3.8875656e-05, 2.1865431e-05, 7.5938424e-06, 1.1002024e-05, 1.4697852e-05, 1.0363219e-05, 1.3873483e-05, 5.480308e-0
6, 2.5079515e-05, 1.880709e-05, 3.36869e-05, 4.963569e-06, 5.0399573e-05, 8.038155e-06, 1.2524423e-05, 8.9115665e-06, 2.6291995
e-05, 1.2842291e-05, 1.670077e-05, 8.102939e-06, 1.7831999e-05, 1.23781365e-05, 1.8762637e-05, 1.5174702e-05, 6.845772e-06, 1.1
689196e-05, 1.2800766e-05, 1.0450647e-05, 1.4097009e-05, 1.6386899e-05, 2.0247615e-05, 8.705659e-06, 2.4991865e-05, 1.5830783e-
05, 1.3842462e-05, 2.5903151e-05, 1.3370266e-05, 1.1833138e-05, 2.9003275e-05, 1.4443052e-05, 1.854564e-05, 7.4380205e-06, 1.44
98808e-05, 6.77165e-06, 2.885395e-05, 1.563785e-05, 3.1643005e-05, 3.2899556e-05, 1.4158686e-05, 1.4767217e-05, 1.770437e-05,
1.9277742e-05, 2.087986e-05, 9.996074e-06, 2.7425942e-05, 1.8587909e-05, 1.9921108e-05, 1.3307672e-05, 1.1244466e-05, 1.7869537
e-05, 2.5420935e-05, 1.7581044e-05, 1.1064866e-05, 1.895345e-05, 9.253465e-06, 1.6730983e-05, 2.7721753e-05, 1.2695167e-05, 1.0
866935e-05, 2.9413626e-05, 1.20990735e-05, 1.7110817e-05, 4.2601663e-05, 2.798919e-05, 1.2015782e-05, 2.1526386e-05, 1.37411525
e-05, 2.7254899e-05, 2.042989e-05, 2.54647e-05, 1.549725e-05, 1.982815e-05]
PROM_MSE_45
1.7976165950130962e-05
DESV_MSE_45
8.476037e-06
```

```
In [53]: Lista_MSE_50 = Iteration(50)
PROM_MSE_50 = sum(Lista_MSE_50)/len(Lista_MSE_50)
DESV_MSE_50 = np.std(Lista_MSE_50)
print('Lista_MSE_50')
print(Lista_MSE_50)
print('PROM_MSE_50')
print(PROM_MSE_50)
print('DESV_MSE_50')
print(DESV_MSE_50)
```

```
Lista_MSE_50
[2.1693724e-05, 1.2858481e-05, 9.071983e-06, 2.5213796e-05, 2.7692768e-05, 1.07946535e-05, 1.5793072e-05, 2.8818284e-05, 2.1312
533e-05, 1.655944e-05, 9.3018225e-06, 2.3982246e-05, 2.5042807e-05, 1.3345496e-05, 5.530639e-06, 1.6595888e-05, 2.6914853e-05,
1.1134577e-05, 1.1774143e-05, 2.7938195e-05, 1.2960337e-05, 6.7065243e-06, 1.50408405e-05, 6.7140813e-06, 4.081307e-06, 1.46248
34e-05, 1.7852128e-05, 5.143957e-06, 8.183694e-06, 1.2287336e-05, 1.5055222e-05, 1.4241759e-05, 1.0905387e-05, 9.934902e-06, 1.
6765165e-05, 1.1999164e-05, 1.093562e-05, 1.125754e-05, 1.2132418e-05, 1.7079092e-05, 3.0002293e-05, 1.6325392e-05, 1.1285829e-
05, 2.5252186e-05, 2.1776026e-05, 1.1355616e-05, 2.422439e-05, 1.1072117e-05, 1.0638124e-05, 1.2650756e-05, 5.7993543e-06, 2.30
57446e-05, 1.3413933e-05, 2.945718e-05, 1.2218257e-05, 2.394525e-05, 3.0293866e-05, 1.1956879e-05, 4.103407e-05, 5.713178e-06,
1.642317e-05, 2.7764756e-05, 1.1060569e-05, 1.2348787e-05, 9.124587e-06, 9.694514e-06, 2.2613118e-05, 1.2841409e-05, 2.4998502e
-05, 2.2069313e-05, 1.9722584e-05, 2.2845868e-05, 1.6460013e-05, 1.6241369e-05, 3.5234632e-05, 8.120963e-06, 1.500081e-05, 1.02
66747e-05, 2.4577288e-05, 1.0811309e-05, 8.671405e-06, 1.2916276e-05, 1.7176713e-05, 1.2031651e-05, 1.5300539e-05, 4.8449216e-0
5, 9.99849e-06, 1.2783439e-05, 1.9031557e-05, 1.9017249e-05, 2.4932853e-05, 1.0447075e-05, 7.67839e-06, 1.4997247e-05, 7.265603
5e-06, 3.7701913e-05, 1.5355196e-05, 7.1148024e-06, 1.5528165e-05, 1.8488454e-05]
PROM_MSE_50
1.649849388286384e-05
DESV_MSE_50
8.201067e-06
```

Las listas se crean para cada cierto número de neuronas obteniendo los siguientes resultados.

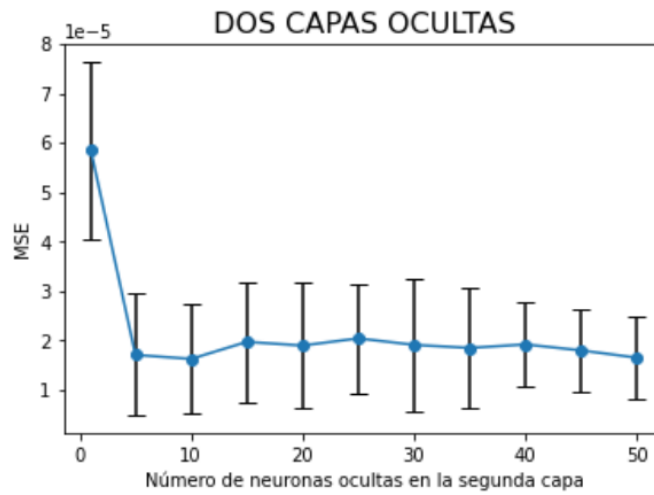
```
In [54]: PROM = [PROM_MSE_1,PROM_MSE_5,PROM_MSE_10,PROM_MSE_15,PROM_MSE_20,PROM_MSE_25,PROM_MSE_30,PROM_MSE_35,PROM_MSE_40,PROM_MSE_45,PROM_MSE_50]
DESV = [DESV_MSE_1,DESV_MSE_5,DESV_MSE_10,DESV_MSE_15,DESV_MSE_20,DESV_MSE_25,DESV_MSE_30,DESV_MSE_35,DESV_MSE_40,DESV_MSE_45,DESV_MSE_50]
```

```
In [55]: print(PROM)
print(DESV)

[5.848956900081248e-05, 1.706354720681702e-05, 1.624330882123104e-05, 1.9697220036505314e-05, 1.8947345108699664e-05, 2.0408715
24513932e-05, 1.909119779156754e-05, 1.8468749581188603e-05, 1.9180938143108505e-05, 1.7976165950130962e-05, 1.649849388286384e
-05]
[1.7966402e-05, 1.2344372e-05, 1.1073417e-05, 1.21101375e-05, 1.2654298e-05, 1.1104764e-05, 1.3528145e-05, 1.2121503e-05, 8.653
03e-06, 8.476037e-06, 8.201067e-06]
```

Realizando siguiente gráfico con la librería MATPLOTLIB.

```
In [4]: #Gráfico
plt.xlabel('Número de neuronas ocultas en la segunda capa')
plt.ylabel('MSE')
plt.title('DOS CAPAS OCULTAS',fontsize=16)
X_Neurons=[1,5,10,15,20,25,30,35,40,45,50]
MSE=PROM
yerror=DESV
plt.errorbar(X_Neurons,MSE,yerr=yerror,ecolor="black", marker='o',capsize=5)
plt.show()
```



Los datos obtenidos del simulador y los códigos con los que se realizó la metodología se pueden encontrar a continuación:

<https://github.com/AbrahamIQRodarte/Columna-pared.git>

CAPÍTULO 9

REFERENCIAS

9.1 REFERENCIAS

- Aslan, N., Ozmen Koca, G., Kobat, M.A., Dogan, S., 2022. Multi-classification deep CNN model for diagnosing COVID-19 using iterative neighborhood component analysis and iterative ReliefF feature selection techniques with X-ray images. *Chemometrics and Intelligent Laboratory Systems* 224, 104539. <https://doi.org/10.1016/j.chemolab.2022.104539>
- Azlan Hussain, M., 1999. Review of the applications of neural networks in chemical process control — simulation and online implementation. *Artificial Intelligence in Engineering* 13, 55–68. [https://doi.org/10.1016/S0954-1810\(98\)00011-9](https://doi.org/10.1016/S0954-1810(98)00011-9)
- Behrad, F., Saniee Abadeh, M., 2022. An overview of deep learning methods for multimodal medical data mining. *Expert Syst Appl* 200, 117006. <https://doi.org/10.1016/j.eswa.2022.117006>
- Cardozo, N., Mens, K., 2022. Programming language implementations for context-oriented self-adaptive systems. *Inf Softw Technol* 143, 106789. <https://doi.org/10.1016/j.infsof.2021.106789>
- Chang, Z., Zhang, Y., Chen, W., 2019. Electricity price prediction based on hybrid model of adam optimized LSTM neural network and wavelet transform. *Energy* 187. <https://doi.org/10.1016/j.energy.2019.07.134>
- Cheng, S., Wu, Y., Li, Y., Yao, F., Min, F., 2021. TWD-SFNN: Three-way decisions with a single hidden layer feedforward neural network. *Inf Sci (N Y)* 579, 15–32. <https://doi.org/10.1016/j.ins.2021.07.091>
- Conte, E., Pierrri, G., Federici, A., Mendolicchio, L., Zbilut, J.P., 2006. A model of biological neuron with terminal chaos and quantum-like features. *Chaos Solitons Fractals* 30, 774–780. <https://doi.org/10.1016/j.chaos.2005.08.211>
- Cooper, S.J., 2005. Donald O. Hebb's synapse and learning rule: a history and commentary. *Neurosci Biobehav Rev* 28, 851–874. <https://doi.org/10.1016/j.neubiorev.2004.09.009>
- de Araújo Neto, A.P., Sales, F.A., Brito, R.P., 2021. Controllability comparison for extractive dividing-wall columns: ANN-based intelligent control system versus conventional control system. *Chemical Engineering and Processing - Process Intensification* 160, 108271. <https://doi.org/10.1016/j.cep.2020.108271>
- Deng, H.-F., Sun, M.-W., Wang, Y., Zeng, J., Yuan, T., Li, T., Li, D.-H., Chen, W., Zhou, P., Wang, Q., Jiang, H., 2022. Evaluating machine learning models for sepsis prediction: A systematic review of methodologies. *iScience* 25, 103651. <https://doi.org/10.1016/j.isci.2021.103651>

- Domínguez Mayorga, C.R., Espejel Rivera, M.A., Ramos Velasco, L.E., Ramos Fernández, J.C., Escamilla Hernández, E., 2012. Algoritmos Wavenet con Aplicaciones en la Aproximación de Señales: un Estudio Comparativo. *Revista Iberoamericana de Automática e Informática Industrial RIAI* 9, 347–358.
<https://doi.org/10.1016/j.riai.2012.09.001>
- El-Gendy, E.M., Saafan, M.M., Elksas, M.S., Saraya, S.F., Areed, F.F.G., 2019. New Suggested Model Reference Adaptive Controller for the Divided Wall Distillation Column. *Ind Eng Chem Res* 58, 7247–7264. <https://doi.org/10.1021/acs.iecr.9b01747>
- Errico, M., Sanchez-Ramirez, E., Quiroz-Ramirez, J.J., Rong, B.-G., Segovia-Hernandez, J.G., 2017. Multiobjective Optimal Acetone–Butanol–Ethanol Separation Systems Using Liquid–Liquid Extraction-Assisted Divided Wall Columns. *Ind Eng Chem Res* 56. <https://doi.org/10.1021/acs.iecr.7b03078>
- Gómez-Castro, F.I., Segovia-Hernández, J.G., Hernández, S., Gutiérrez-Antonio, C., Briones-Ramírez, A., 2008. Dividing Wall Distillation Columns: Optimization and Control Properties. *Chem Eng Technol* 31. <https://doi.org/10.1002/ceat.200800116>
- González-Bravo, R., Sánchez-Ramírez, E., Quiroz-Ramírez, J.J., Segovia-Hernández, J.G., Lira-Barragán, L.F., Ponce-Ortega, J.M., 2016. Total Heat Integration in the Biobutanol Separation Process. *Ind Eng Chem Res* 55.
<https://doi.org/10.1021/acs.iecr.5b03168>
- Graham-Cumming, J., 2012. Alan Turing: Intelligence & life. *New Sci* (1956) 214, vi–vii. [https://doi.org/10.1016/S0262-4079\(12\)61378-5](https://doi.org/10.1016/S0262-4079(12)61378-5)
- Han, J., Kang, S., 2022. Dynamic imputation for improved training of neural network with missing values. *Expert Syst Appl* 194, 116508.
<https://doi.org/10.1016/j.eswa.2022.116508>
- Hewett, T.T., Marcus, M., 2002. The case for a Benjamin Franklin Medal in Computer and Cognitive Science for Marvin L. Minsky. *J Franklin Inst* 339, 295–302.
[https://doi.org/10.1016/S0016-0032\(01\)00045-X](https://doi.org/10.1016/S0016-0032(01)00045-X)
- Hochreiter, S., Schmidhuber, J., 1997. Long Short-Term Memory. *Neural Comput* 9.
<https://doi.org/10.1162/neco.1997.9.8.1735>
- Hüsken, M., Stagge, P., 2003. Recurrent neural networks for time series classification. *Neurocomputing* 50. [https://doi.org/10.1016/S0925-2312\(01\)00706-8](https://doi.org/10.1016/S0925-2312(01)00706-8)
- Kakkar, S., Kwapinski, W., Howard, C.A., Kumar, K.V., 2021. Deep neural networks in chemical engineering classrooms to accurately model adsorption equilibrium data. *Education for Chemical Engineers* 36, 115–127.
<https://doi.org/10.1016/j.ece.2021.04.003>
- Keenan, T.D.L., Chen, Q., Agrón, E., Tham, Y.-C., Goh, J.H.L., Lei, X., Ng, Y.P., Liu, Y., Xu, X., Cheng, C.-Y., Bikbov, M.M., Jonas, J.B., Bhandari, S., Broadhead, G.K.,

- Colyer, M.H., Corsini, J., Cousineau-Krieger, C., Gensheimer, W., Grasic, D., Lamba, T., Magone, M.T., Maiberger, M., Oshinsky, A., Purt, B., Shin, S.Y., Thavikulwat, A.T., Lu, Z., Chew, E.Y., Ajilore, P., Akman, A., Azar, N.S., Azar, W.S., Chan, B., Cox, V., Dave, A.D., Dhanjal, R., Donovan, M., Farrell, M., Finkel, F., Goblirsch, T., Ha, W., Hill, C., Kumar, A., Kent, K., Lee, A., Patel, P., Peprah, D., Piliponis, E., Selzer, E., Swaby, B., Tenney, S., Zeleny, A., 2022. DeepLensNet: Deep Learning Automated Diagnosis and Quantitative Classification of Cataract Type and Severity. *Ophthalmology*. <https://doi.org/10.1016/j.optha.2021.12.017>
- Khodayar, M., Wang, J., 2019. Spatio-Temporal Graph Deep Neural Network for Short-Term Wind Speed Forecasting. *IEEE Trans Sustain Energy* 10. <https://doi.org/10.1109/TSTE.2018.2844102>
- KLIR, G.J., 1997. A review of: “ *INDUSTRIAL INTELLIGENT CONTROL: Fundamentals and Applications* ” by Yong-Zai Lu. John Wiley, Chichester (U.K.) and New York, 1996. XX 325 pages, ISBN 0-471-95058-0. *Int J Gen Syst* 26, 291–291. <https://doi.org/10.1080/03081079708945183>
- Kumari, P., Toshniwal, D., 2021. Deep learning models for solar irradiance forecasting: A comprehensive review. *J Clean Prod* 318. <https://doi.org/10.1016/j.jclepro.2021.128566>
- Lanska, D.J., 2014. Lashley, Karl Spencer, in: *Encyclopedia of the Neurological Sciences*. Elsevier, pp. 842–843. <https://doi.org/10.1016/B978-0-12-385157-4.00939-8>
- Lee, S.-K., Lee, H., Back, J., An, K., Yoon, Y., Yum, K., kim, S., Hwang, S.-U., 2021. Prediction of tire pattern noise in early design stage based on convolutional neural network. *Applied Acoustics* 172, 107617. <https://doi.org/10.1016/j.apacoust.2020.107617>
- Liu, D., Kong, H., Luo, X., Liu, W., Subramaniam, R., 2022. Bringing AI to edge: From deep learning’s perspective. *Neurocomputing* 485, 297–320. <https://doi.org/10.1016/j.neucom.2021.04.141>
- Liu, Y., Wang, Z., Ma, Q., Shen, H., 2022. Multistability analysis of delayed recurrent neural networks with a class of piecewise nonlinear activation functions. *Neural Networks*. <https://doi.org/10.1016/j.neunet.2022.04.015>
- Lu, J., Wang, Q., Zhang, Z., Tang, J., Cui, M., Chen, X., Liu, Q., Fei, Z., Qiao, X., 2021. Surrogate modeling-based multi-objective optimization for the integrated distillation processes. *Chemical Engineering and Processing - Process Intensification* 159. <https://doi.org/10.1016/j.cep.2020.108224>
- Manickam, A., Jiang, J., Zhou, Y., Sagar, A., Soundrapandiyan, R., Dinesh Jackson Samuel, R., 2021. Automated pneumonia detection on chest X-ray images: A deep learning approach with different optimizers and transfer learning architectures. *Measurement* 184. <https://doi.org/10.1016/j.measurement.2021.109953>

- Moon, J., Gbadago, D.Q., Hwang, G., Lee, D., Hwang, S., 2022. Software platform for high-fidelity-data-based artificial neural network modeling and process optimization in chemical engineering. *Comput Chem Eng* 158, 107637. <https://doi.org/10.1016/j.compchemeng.2021.107637>
- Neves, T.G., de Araújo Neto, A.P., Sales, F.A., Vasconcelos, L.G.S., Brito, R.P., 2021. ANN-based intelligent control system for simultaneous feed disturbances rejection and product specification changes in extractive distillation process. *Sep Purif Technol* 259, 118104. <https://doi.org/10.1016/j.seppur.2020.118104>
- Nguyen, N.D., Nguyen, V.T., 2022. Development of ANN structural optimization framework for data-driven prediction of local two-phase flow parameters. *Progress in Nuclear Energy* 146, 104176. <https://doi.org/10.1016/j.pnucene.2022.104176>
- Ookura, S., Mori, H., 2020. An Efficient Method for Wind Power Generation Forecasting by LSTM in Consideration of Overfitting Prevention. *IFAC-PapersOnLine* 53. <https://doi.org/10.1016/j.ifacol.2020.12.1008>
- Pereira, L.G., Chagas, M.F., Dias, M.O.S., Cavalett, O., Bonomi, A., 2015. Life cycle assessment of butanol production in sugarcane biorefineries in Brazil. *J Clean Prod* 96, 557–568. <https://doi.org/10.1016/j.jclepro.2014.01.059>
- Portet, S., 2020. A primer on model selection using the Akaike Information Criterion. *Infect Dis Model* 5, 111–128. <https://doi.org/10.1016/j.idm.2019.12.010>
- Poznyak, A., Chairez, I., Poznyak, T., 2019. A survey on artificial neural networks application for identification and control in environmental engineering: Biological and chemical systems with uncertain models. *Annu Rev Control* 48, 250–272. <https://doi.org/10.1016/j.arcontrol.2019.07.003>
- Ricardo, B.-M.J., Rafael, V.-R., 2015. Algoritmo que ejecuta submodelos de un modelo matemático para mayor eficiencia. *Ingeniería, Investigación y Tecnología* 16, 551–563. <https://doi.org/10.1016/j.riit.2015.09.007>
- Rogers, S., 1997. Adaptive inverse control. *Control Eng Pract* 5, 146–147. [https://doi.org/10.1016/S0967-0661\(97\)89242-8](https://doi.org/10.1016/S0967-0661(97)89242-8)
- Seidel, T., Ränger, L.-M., Grützner, T., Bortz, M., 2022. Simultaneous simulation and optimization of multiple dividing wall columns. *Comput Chem Eng* 157, 107607. <https://doi.org/10.1016/j.compchemeng.2021.107607>
- Shin, Y., Smith, R., Hwang, S., 2020a. Development of model predictive control system using an artificial neural network: A case study with a distillation column. *J Clean Prod* 277. <https://doi.org/10.1016/j.jclepro.2020.124124>
- Shin, Y., Smith, R., Hwang, S., 2020b. Development of model predictive control system using an artificial neural network: A case study with a distillation column. *J Clean Prod* 277, 124124. <https://doi.org/10.1016/j.jclepro.2020.124124>

- Srinivasan, M., 2022. Design and manufacture of graphite components for 21st century small modular reactors. *Nuclear Engineering and Design* 386, 111568. <https://doi.org/10.1016/j.nucengdes.2021.111568>
- Tian, S., Arshad, N.I., Toghraie, D., Eftekhari, S.A., Hekmatifar, M., 2021. Using perceptron feed-forward Artificial Neural Network (ANN) for predicting the thermal conductivity of graphene oxide-Al₂O₃/water-ethylene glycol hybrid nanofluid. *Case Studies in Thermal Engineering* 26, 101055. <https://doi.org/10.1016/j.csite.2021.101055>
- Tian, Y., Meduri, V., Bindlish, R., Pistikopoulos, E.N., 2022. A Process Intensification synthesis framework for the design of dividing wall column systems. *Comput Chem Eng* 160, 107679. <https://doi.org/10.1016/j.compchemeng.2022.107679>
- Tsai, J.-M., Chien, H.-H., Shih, S.-C., Lee, S.-C., Tsai, L.-Y., Tsay, S.-L., 2017. Using Balanced Scorecard on Reducing Fall Incidents and Injuries Among Elderly Cancer Patients in a Medical Center in Taiwan. *Int J Gerontol* 11. <https://doi.org/10.1016/j.ijge.2016.05.012>
- Vadakara, J., Basu, S., 2019. Python Tools for Stem Cell Transplantation. *Biology of Blood and Marrow Transplantation* 25. <https://doi.org/10.1016/j.bbmt.2018.12.209>
- Vepa, R., 2009. Modelling and Estimation of Chaotic Biological Neurons. *IFAC Proceedings Volumes* 42, 27–32. <https://doi.org/10.3182/20090622-3-UK-3004.00008>
- Villegas-Urbe, C.A., Medina-Herrera, N., Hernández-Magallanes, J.A., Tututi-Avila, S., 2022. Optimal design and control of three simplified sargent four-product dividing-wall columns. *Chemical Engineering and Processing - Process Intensification* 174, 108860. <https://doi.org/10.1016/j.cep.2022.108860>
- Wang, Z., 2022. Use of supervised machine learning to detect abuse of COVID-19 related domain names. *Computers and Electrical Engineering* 100, 107864. <https://doi.org/10.1016/j.compeleceng.2022.107864>
- Wilson, E.A., 2008. Affect, artificial intelligence, and internal space. *Emot Space Soc* 1, 22–27. <https://doi.org/10.1016/j.emospa.2008.08.002>
- Yadav, R.K., Anubhav, 2020. PSO-GA based hybrid with Adam Optimization for ANN training with application in Medical Diagnosis. *Cogn Syst Res* 64, 191–199. <https://doi.org/10.1016/j.cogsys.2020.08.011>
- Yan, Z., Chen, J., Hu, R., Huang, T., Chen, Y., Wen, S., 2020. Training memristor-based multilayer neuromorphic networks with SGD, momentum and adaptive learning rates. *Neural Networks* 128. <https://doi.org/10.1016/j.neunet.2020.04.025>
- Yildirim, Ö., Kiss, A.A., Kenig, E.Y., 2011. Dividing wall columns in chemical process industry: A review on current activities. *Sep Purif Technol* 80, 403–417. <https://doi.org/10.1016/j.seppur.2011.05.009>

Zetterberg, H., 2018. Tauomics and Kinetics in Human Neurons and Biological Fluids. Neuron 97, 1202–1205. <https://doi.org/10.1016/j.neuron.2018.02.030>

Zhang, G., Eddy Patuwo, B., Y. Hu, M., 1998. Forecasting with artificial neural networks: Int J Forecast 14, 35–62. [https://doi.org/10.1016/S0169-2070\(97\)00044-7](https://doi.org/10.1016/S0169-2070(97)00044-7)