Aprendizaje de Máquina Aplicado al Control

Autor

Daniel Felipe Rambaut Lemus

Trabajo presentado como requisito para optar por el
título de Máster en Matemáticas Aplicadas y Ciencias de la Computación

Director

Germán Obando

Escuela de Ingeniería Ciencia y Tecnología
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

Bogotá-Colombia

2023

Agradecimientos

Quiero expresar mi profundo agradecimiento a aquellos que han estado a mi lado durante este viaje. En primer lugar, agradezco a mi familia por su incansable apoyo y amor. Gracias por creer en mí y animarme a seguir adelante, incluso cuando me sentía abrumado. Este logro es tanto mío como de ustedes y estoy agradecido de tenerlos a mi lado. También quiero agradecer al profesor Germán por su guía, conocimiento y paciencia. Gracias por compartir su sabiduría y por ayudarme a desarrollar mis habilidades y conocimientos. No podría haber completado esta tesis sin su dirección y apoyo. Además, quiero agradecer a Andrés Salazar por su amistad, consejos y apoyo incondicional. Gracias por escucharme, motivarme y ayudarme a mantenerme enfocado en mis objetivos. Por último, pero no menos importante, quiero agradecer a mi novia Katherin por su amor, apoyo y paciencia. Gracias por ser mi compañera de vida y por ayudarme a superar cualquier obstáculo que se presentara en mi camino. No podría haber completado esta tesis sin su amor y motivación constante.

Espero que estos agradecimientos ayuden a expresar mi gratitud hacia aquellos que me ayudaron a desarrollar esta tesis de maestría.

Resumen

El objetivo de este trabajo es emular la acción de un controlador utilizando modelos de inteligencia artificial (IA). Para ello, se empleó como planta un sistema de segundo orden que describe la temperatura en un cuarto. Sobre dicha planta, se diseña un controlador predictivo basado en modelo (MPC, por sus siglas en inglés) como referencia para entrenar los algoritmos de IA. MPC es un método que utiliza modelos matemáticos para predecir el comportamiento futuro del sistema y tomar acciones de control óptimas en función de ciertos objetivos preestablecidos.

La emulación del controlador puede plantearse como un problema de regresión, por lo tanto se emplearon tres de los modelos más populares de IA para efectuar regresiones: regresión lineal, vectores de soporte y redes neuronales. Para el entrenamiento de los modelos de IA, se utilizó una base de datos generada al simular el comportamiento del controlador MPC sobre la planta de temperatura.

Se realizaron diferentes pruebas para evaluar el desempeño de los modelos de IA comparándolos con el controlador MPC. Los resultados mostraron que los modelos de IA pueden ser utilizados con éxito para emular dicho controlador con la ventaja de tener un menor costo computacional. En este sentido, cabe resaltar que MPC necesita resolver iterativamente un problema de optimización, mientras que los algoritmos de IA usados sólo requieren evaluar cierta función (que se obtiene al entrenar los modelos) en cada iteración de control.

En conclusión, esta investigación es un primer paso exitoso en un camino prometedor: el uso de IA para el control de procesos dinámicos.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

El aprendizaje automático (ML por sus siglas en inglés) [12] es una rama de la Inteligencia Artificial (IA). ML proporciona a la máquina la capacidad de aprender y tomar decisiones, dos habilidades que caracterizan a los seres humanos. El hecho de usar algoritmos para realizar estas dos tareas no es nuevo, ya que tan pronto como aparecieron las nuevas tecnologías, se desarrollaron métodos sistemáticos que permiten cumplir estas acciones. Sin embargo, durante las últimas décadas, el impulso que ha tenido ML ha sido tan fuerte que se ha convertido en el mayor foco de desarrollo de los últimos años. Su uso ha revolucionado todo tipo de campos en ciencia [26], ingeniería [22], medicina [9], entre otros. El procesamiento de imágenes, el reconocimiento de patrones, la extracción de texto, el reconocimiento de voz y la traducción automática se han beneficiado considerablemente de este desarrollo [16].

La teoría de control examina el diseño acciones que llevan a los sistemas dinámicos a comportarse siguiendo unos lineamientos [1]. En particular, dentro de esta teoría encontramos los algoritmos de control óptimo que, por su naturaleza, han mostrado tener buenos desempeños en una gran variedad de aplicaciones [14]. Uno de los algoritmos de control óptimo más reconocidos es el Control Predictivo basado en Modelo (MPC por sus siglas en inglés) [3]. Sin embargo, es bien sabido que el cálculo numérico es la principal barrera para poner en práctica esta técnica. Este obstáculo se acentúa cuando se abordan problemas de dimensión alta (es decir, cuando el número de variables de estado es elevado). Desafortunadamente, hoy en día los problemas de control [10] crecen en dimensión de forma continua, limitando

fuertemente el uso de MPC.

Es natural que se consideren las nuevas posibilidades que ofrece ML para superar el desafío de reducir la carga computacional de los algoritmos de MPC. Esto explica por qué, en los últimos años, hemos sido testigos de resultados innovadores que fusionan las áreas de control óptimo y ML. Hablando a nivel general, podemos dividir estos trabajos en dos categorías: teoría de control para aprendizaje automático [6], [7], [8] y aprendizaje automático para teoría de control [15], [20], [27]. La primera se refiere al uso de la teoría de control como una herramienta matemática para formular y resolver problemas teóricos y prácticos de aprendizaje automático, como el ajuste óptimo de parámetros y el entrenamiento de redes neuronales; mientras que la segunda explora el cómo usar prácticas de aprendizaje automático (como el método kernel, o ciertos modelos de regresión) dentro de los algoritmos de control.

Hay, de hecho, muchas pruebas que respaldan el argumento de las estrechas conexiones entre el aprendizaje automático y la teoría del control. Un ejemplo típico es el aprendizaje por refuerzo (RL por sus siglas en inglés) [24], que estudia cómo usar datos pasados para mejorar la manipulación futura de un sistema dinámico. En [20] se describe un esfuerzo reciente por unificar RL con la teoría de control.

La contribución que presentamos en este artículo se centra en la fusión de IA con control óptimo. Específicamente, en nuestro trabajo emulamos las acciones de un controlador utilizando modelos de ML. Para ello, empleamos como planta un sistema de segundo orden que describe la temperatura de un recinto. Sobre dicha planta, se diseña un MPC que se usa como referencia para entrenar los algoritmos de ML.

Mostramos que la emulación del controlador puede plantearse como un problema de regresión. En consonancia con esta propuesta, empleamos tres de los modelos más populares de ML para efectuar regresiones: regresión lineal, vectores de soporte y redes neuronales. Para el entrenamiento de los modelos de ML mencionados,

se usa una base de datos generada al simular el comportamiento del controlador MPC sobre la planta de temperatura mencionada.

A través de simulaciones, realizamos diferentes pruebas para evaluar el desempeño de los modelos de IA comparados con los resultados obtenidos al usar el controlador MPC. Los experimentos mostraron que los modelos de IA pueden ser utilizados con éxito para emular dicho controlador con la ventaja añadida de tener un menor costo computacional.

2 | Preliminares

2.1. Modelo del sistema a controlar

En esta sección estudiaremos la aplicación de MPC para sistemas lineales generales en tiempo discreto. El modelo del sistema lineal en tiempo discreto que estudiaremos tiene la siguiente estructura.

$$x_{k+1} = Ax_k + Bu_k \tag{1}$$

Donde $k \in \mathbf{N}_0$ es el índice de tiempo, $x_k \in \mathbb{R}^n$ son los estados del sistema, $u_k \in \mathbb{R}^m$ las entradas de control, y $A \in \mathbb{R}^{nxn}$, $B \in \mathbb{R}^{nxm}$ son respectivamente las matrices del sistema y la matriz de entradas [2]. Para los estados y las entradas, se definen las siguientes restricciones

$$x_k \in \chi \ , \ u_k \in U \quad \forall k \in \mathbb{N}_0,$$

El conjunto de estados χ es cerrado y U es compacto, ambos conjuntos son convexos y contienen al origen.

2.2. Formulación del MPC

MPC es un método que iterativamente ejerce sobre el sistema a controlar una acción que minimiza cierta función objetivo. Dicha minimización se efectúa dentro de una ventana de tiempo conocida como horizonte de predicción. La función objetivo se define en términos de variables presentes y pronosticadas, que se obtienen a través de un modelo explícito del sistema a controlar. Es importante anotar que al aplicar MPC debe resolverse un problema de optimización para encontrar la acción de control que será aplicada al sistema. Este procedimiento se repite en cada iteración del controlador. Es decir, cada vez que transcurra un intervalo de tiempo fijo conocido como tiempo de muestreo. Una ventaja de MPC es que puede considerar restricciones en los estados y en las entradas de control, las cuales se incluyen de forma natural en la formulación del problema de optimización mencionado. Cabe mencionar que MPC puede ser aplicado a sistemas lineales o no lineales [23], en esta tesis nos centramos en la aplicación de MPC en sistemas lineales.

En el problema de optimización embebido dentro del MPC, los estados del sistema se predicen con la Ecuación 1. A continuación, la predicción de los estados en los i-ésimos pasos de tiempo después del tiempo actual k se denotan como $\hat{x}_{k+i|k}$. Las predicciones se basan en el estado medido actual del sistema 1, que es el k-ésimo estado x_k, es decir, $\hat{x}_{k|k} = x_k$. Con base en este estado, se calcula una secuencia de entradas de control. La i-ésima posición de dicha secuencia se denota como $\hat{u}_{k+i|k}$.

El objetivo del controlador es dirigir los estados al origen. Para tal fin, se emplea una función objetivo que considera las predicciones en los próximos N pasos. Así, N es el horizonte de predicción del problema de optimización. Por lo general, para los estados y las entradas se usa una función de costo cuadrática

$$\mathbb{J}(\hat{x}_k|U_k) = x^T\theta + u^T R u$$

Ya que existen métodos eficientes de optimización cuadrática [18]. La matriz de pesos para los estados se denota como $Q \in \mathbb{R}^{n\times n}$ y es semidefinida positiva, es decir, $Q \succeq 0$. La matriz de pesos para las entradas es $R \in \mathbb{R}^{m\times m}$ y es definida positiva, es decir, $R \succ 0$. Ambas matrices son dadas por el problema y normalmente se eligen como matrices diagonales. Un valor alto de Q conduce a una convergencia más rápida del estado correspondiente, mientras que un valor alto de R reduce la amplitud de la entrada. Por lo tanto, el problema de optimización del MPC está definido como

$$\begin{aligned}
U_k^* = \quad & arg\min_{U_k} \mathbb{J}(\hat{x}_{k|k}|U_k) \\
= \quad & arg\min_{U_k} \sum_{i=0}^{N-1} [\hat{x}_{k+i|k}^T Q \hat{x}_{k+i|k} + \hat{u}_{k+i|k} R \hat{u}_{k+i|k}] + \hat{x}_{k+N|k}^T Q_f \hat{x}_{k+N|k}^T \\
\text{s.t.} \quad & \hat{x}_{k+i+1|k} = A\hat{x}_{k+i|k} + B\hat{u}_{k+i|k} \;\; i \in \mathbb{N}_{0:N-1} \\
& \hat{u}_{k+i|k} \in U \;\; i \in \mathbb{N}_{0:N-1} \\
& \hat{x}_{k+i|k} \in \chi \;\; i \in \mathbb{N}_{0:N-1} \\
& \hat{x}_{k+N|k} \in \chi_f \;\; i \in \mathbb{N}_{0:N-1}
\end{aligned} \tag{2}$$

Donde $N_{a:b}$ denota el conjunto de todos los números enteros entre a y b, incluyéndolos. La optimización minimiza la función de costo con respecto a todas las entradas de control previstas. Por lo tanto, todas las entradas se combinan en un vector de secuencia de entradas:

$$U_k = \begin{pmatrix} \hat{u}_{k|k} \\ \hat{u}_{k+1|k} \\ \vdots \\ \hat{u}_{k+N-1|k} \end{pmatrix} \in U^N \subset \mathbb{R}^{mN} \tag{3}$$

Donde U_k^* es la solución óptima para la secuencia de entrada. Además, en 2 se usa el costo terminal $Q_f \in \mathbb{R}^{n\times n}$ para garantizar la estabilidad. De no ser así, es posible que la factibilidad recursiva, que es la propiedad de MPC, que asegura que el problema de optimización tenga solución en cada paso de tiempo, será afectada, y por tanto el controlador falle.

Basado en $\hat{x}_{k|k}$ el problema optimización (2) se aplica para obtener la secuencia de control óptima U_k^*. La ley de control para el tiempo actual es entonces el primer elemento de la secuencia de control óptima, es decir, $u_k = k(x_k) = \hat{u}_{k|k}^*$. Esta acción de control es la que se aplica al sistema (las acciones restantes no se usan).

Todas las entradas restantes no se utilizan en la ley de control. Esta entrada u_k se aplica al sistema 1. En el siguiente paso $k+1$ se repite el procedimiento en base a la medida del estado subsiguiente x_{k+1}.

Finalmente, consideremos algunas ventajas que trae utilizar MPC. Los controladores convencionales solo toman decisiones con respecto a la señal de error actual, pero el controlador MPC también considera las señales de error futuras para tomar una decisión conveniente. A diferencia de otros controladores de retroalimentación que calculan la acción de control en base a información presente o pasada, MPC determina la acción de control en base a la predicción de la dinámica futura del sistema. Debido a la predicción futura, se pueden tomar medidas de control tempranas que tengan en cuenta el comportamiento futuro. MPC puede obtener un mejor rendimiento de control en presencia de restricciones que se prevé se activen en el futuro.

2.3. Inteligencia artificial

La inteligencia artificial (IA) es una forma de hacer que una computadora, una máquina o un software piensen de la misma manera que piensan los seres humanos. Los desarrollos en IA se han logrado estudiando el funcionamiento del cerebro hu-

mano y entendiendo cómo los humanos aprenden, deciden y trabajan mientras intentan resolver un problema. Estos estudios se han tomado como base para el desarrollo de software inteligente [28].

El aprendizaje automático (ML) es una rama de la inteligencia artificial que permite que los sistemas informáticos aprendan directamente de ejemplos, datos y experiencias. Las técnicas de ML se clasifican dependiendo del problema que se quiere resolver en aprendizaje supervisado, no supervisado y por refuerzo. En este trabajo usaremos aprendizaje supervisado.

El **aprendizaje supervisado**, requiere datos etiquetados con las respuestas correctas que se esperan de la salida del algoritmo. Para problemas de clasificación y regresión, el aprendizaje supervisado demostró ser preciso y rápido [4].

1. **Clasificación**: consiste en predecir el valor de salida categórico. En este tipo de problemas se asume que los datos se pueden separar en clases específicas. La clasificación tiene diferentes casos de uso, tales como: determinar el clima o si un correo electrónico es spam o no.

2. **Regresión**: es un tipo de problema en el que se necesita la predicción de un valor continuo, como los precios de las acciones o el precio de la vivienda.

En resumen el aprendizaje supervisado modela las relaciones y dependencias entre la salida de la predicción y las características de entrada, de modo que es posible predecir los valores de salida para datos de entrada dispuestos a los usados en la etapa de entrenamiento del modelo [5].

Hay varios algoritmos de aprendizaje supervisado, como regresión lineal, regresión logística, máquinas de soporte vectorial (svm por sus siglas en inglés), redes neuronales, entre otros. En particular, en esta tesis nos interesa estudiar las regresiones lineales, regresión de vectores de soporte y redes neuronales, ya que serán los modelos de aprendizaje automático que aplicaremos a nuestros datos para sintetizar un controlador que emule el desempeño de un MPC.

2.3.1. Regresión lineal

La regresión lineal ajusta nuestros datos a una función lineal. $y = \beta_0 + \beta_1 x$. Aquí, x se denomina variable independiente o variable predictora y y se denomina variable dependiente o variable respuesta [11]. Antes de hablar sobre cómo realizar el ajuste lineal, demos un vistazo más de cerca a las siguientes variables:

1. β_1: representa la pendiente de la línea, esta es una de las cantidades más importantes en cualquier análisis de regresión lineal. Un valor muy cercano a 0 indica poca relación y valores positivos o negativos grandes representan relaciones positivas o negativas grandes, respectivamente.

2. β_0: representa el intercepto de la línea con el eje vertical.

El mejor ajuste lineal posible es aquel para el cual los datos generados son más probables. Esta es una técnica de uso común en estadística: proponer un modelo probabilístico y usar la probabilidad de los datos para evaluar qué tan bueno es un modelo en particular.

Para comenzar con el modelo probabilístico, observamos puntos de datos emparejados $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, donde se asume que una función de x_i genera cada y_i utilizando alguna línea subyacente y la adición de algún ruido gaussiano, formalmente:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

Aquí, el ruido ε_i representa el hecho de que nuestros datos no se ajustaran perfectamente al modelo. Modelaremos ε_i como $\mathcal{N}(0, \sigma^2)$. Notemos que el intercepto β_0, la pendiente β_1 y la varianza del ruido σ^2 son todos tratados como cantidades fijas (deterministas) pero desconocidas.

Suponiendo que los datos $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ que observamos son generados a través de una función, resulta que podemos encontrar la línea para la cual la

probabilidad de los datos es más alta resolviendo el siguiente problema de optimización.

$$min_{\beta_0,\beta_1} : \sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_i)]^2$$

El problema anterior es conocido como el problema de regresión lineal de mínimos cuadrados. Dado un conjunto de puntos, la solución se define como:

$$\hat{\beta_1} = \frac{\sum_{i=1}^{n} x_i y_i - \frac{1}{n} \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n} (\sum_{i=1}^{n} x_i)^2} \tag{4}$$

$$= r \frac{s_y}{s_x}, \tag{5}$$

$$\hat{\beta_0} = \bar{y} - \hat{\beta_1} \bar{x} \tag{6}$$

Donde $\bar{x}, \bar{y}, s_x, s_y$ son la medias muestrales y las desviaciones estándar para los valores de x, y, respectivamente, y r es el coeficiente de correlación, definido como:

$$r = \frac{1}{n-1} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s_x} \right) \left(\frac{y_i - \bar{y}}{s_y} \right)$$

Finalmente, luego de realizar el ajuste de las constantes en el modelo es posible realizar predicciones para datos nuevos x^*, lo cual se logra con solo reemplazar el dato nuevo en nuestra ecuación $y = \hat{\beta_0} + \hat{\beta_1} x^*$. Cabe resaltar que para el caso de múltiples variables $(x_1, x_2, \ldots, x_n)$ el procedimiento para encontrar el valor de las constantes $\beta_1, \beta_2, \ldots, \beta_n$ tal que $y = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$ es similar al caso descrito anteriormente, con la única diferencia de que los cálculos los tenemos que realizar en n-dimensiones.

2.3.2. Maquinas de soporte vectorial

Las Máquinas de soporte vectorial (SVM por sus siglas en inglés) son un algoritmo de aprendizaje automático supervisado que considera cada característica como un

punto en el espacio n-dimensional y clasifica los datos en dos clases mediante el uso de un hiperplano. Por ejemplo, para un espacio bidimensional, el hiperplano es una línea. En dimensiones superiores (3 y superiores) la separación es un hiperplano [25]. Dos reglas básicas que gobiernan la clasificación mencionada son:

1. Seleccionar el hiperplano que separa los datos en clases distintas.
2. Maximizar la distancia entre los puntos de datos más cercanos de cualquiera de las clases, de modo que ningún punto de datos se clasifique incorrectamente.

SVM se formuló originalmente para abordar problemas de clasificación binaria. De hecho, se utilizó como método para extender la regresión logística para generar modelos de clasificación de máxima entropía. En nuestro estudio, debemos enfocarnos en solucionar un problema de regresión, y por tanto estudiar cómo se comporta SVM para el problema de regresión.

Esbozaremos las ideas detrás de SVM para el problema de regresión. Comenzamos con un conjunto de datos $G = \{(x_i, y_i)\}_{i=1}^{N}$ obtenido por muestreo con ruido $g(x)$. Se requiere determinar una función f tal que aproxime $g(x)$ con base en el conocimiento de G. El SVM para regresión considera funciones de aproximación de la forma:

$$f(x, c) = \sum_{i=1}^{D} c_i \phi_i(x) + b$$

Donde la función $\phi_i(x)$ es llamada función característica, b y c_i son coeficientes que tienen que ser estimados a partir de los datos. Esta forma de aproximación se puede considerar como un hiperplano en el espacio de características D-dimensional definido por las funciones $\phi_i(x)$. Hay que decir que la dimensionalidad del espacio de características no es necesariamente finita. Los coeficientes desconocidos se estiman minimizando la siguiente función:

$$R(c) = \frac{1}{N}\sum_{i=1}^{N} |y_i - f(x_i, c)|_\varepsilon + \lambda||c||^2 \tag{7}$$

Donde λ es una constante y termino $|y_i - f(x_i, c)|_\varepsilon$ se define de la siguiente manera:

$$|y_i - f(x_i, c)|_\varepsilon = \begin{cases} 0 & \text{si } |y_i - f(x_i, c)| < \varepsilon \\ |y_i - f(x_i, c)| - \varepsilon & \text{en caso contrario} \end{cases} \tag{8}$$

Vapnik mostró en [25] que la función que minimiza la Ecuación (7) depende de un número finito de parámetros y tiene la siguiente forma:

$$f(x, \alpha, \alpha^*) = \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) K(x_i, x_i) + b$$

Donde $\alpha_i^* \alpha_i = 0$, $\alpha_i, \alpha_i^* \geq 0$ para $i = 1, \ldots, N$ y $K(x_i, x_i)$ es la llamada función kernel. Esta función describe el producto interno en el espacio de características D-dimensional

$$K(x, y) = \sum_{i=1}^{D} \phi_i(x)\phi_i(y)$$

El hecho interesante es que para muchos conjuntos de funciones $\phi_i(x)$ (incluyendo conjuntos de dimensión infinita) la forma de K es conocida y simple. Además, las características de ϕ_i nunca necesita calcularse en la práctica porque el algoritmo se basa solo en el cómputo de productos escalares en el espacio de características. Varias opciones para el kernel k están disponibles, incluyendo Gaussianas, productos tensoriales, entre otras. Los coeficientes α y α^* son obtenidos maximizando la siguiente forma cuadrática:

$$R(\alpha^*, \alpha) = -\varepsilon \sum_{i=1}^{N}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_i^* - \alpha_i)K(x_i, x_j)$$

Sujetos a las restricciones de $0 \geq \alpha_i^*, \alpha_i \geq C$ y $\sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0$. Debido a la naturaleza de este problema de programación cuadrática, solo un pequeño número de coeficientes $\alpha_i^* - \alpha_i$ serán diferentes de cero, y los puntos de datos asociados con ellos se denominan vectores de soporte. Los parámetros C y ε son dos parámetros libres cuya selección se realiza dependiendo del problema.

2.3.3. Redes Neuronales Artificiales

En esta sección se proporciona una breve presentación de las Redes Neuronales Artificiales (ANN). Una Red Neuronal Artificial es un paradigma de procesamiento de información inspirado en el cerebro humano. Es una simulación biológica realizada en la computadora para efectuar ciertas tareas específicas como agrupación, clasificación y reconocimiento de patrones. Las ANN son aproximadores funcionales universales, tienen formas más generales y flexibles que otros métodos y que son capaces de realizar modelos no lineales sin un conocimiento previo sobre las relaciones entre las variables de entrada y salida.

Las ANN se construyen a partir de un gran número de neuronas o perceptrones. Cada unidad básica puede tomar decisiones simples y transmitir esas decisiones a otras neuronas organizadas en capas interconectadas. Cada una de estas conexiones entre neuronas tiene asociado un peso w, que representa la importancia de la conexión correspondiente en el resultado final. Una red poco profunda tiene solo tres capas de neuronas; una capa de entrada (IL), una capa oculta (HL) y una capa de salida (OL). Mientras que una Red Neuronal Profunda (DNN) tiene dos o más HL. Así DNN son más precisos para resolver problemas complejos [19].

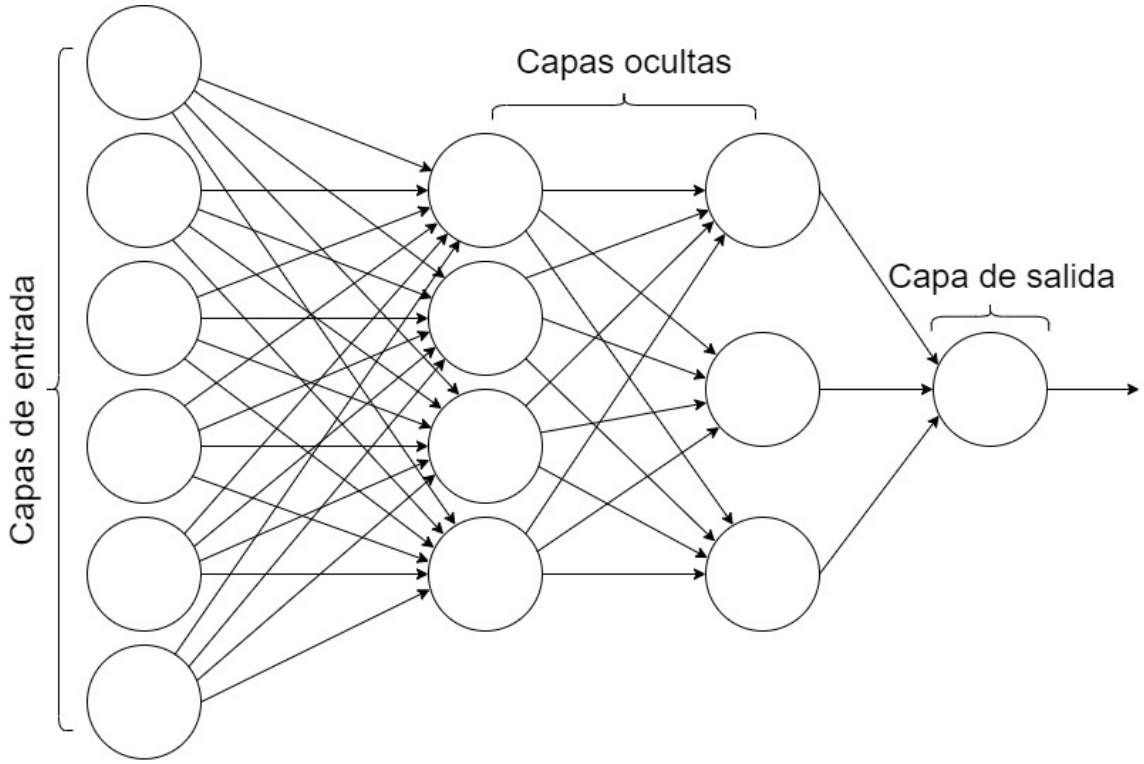

Figura 2.1: Estructura general de una ANN

Para el proceso de aprendizaje de una ANN, en primer lugar, se inicializan los pesos y las entradas x se introducen en la red como primera activación. Luego, cada neurona calcula la suma de las activaciones ponderadas de todas las neuronas en la capa anterior y agrega algo de sesgo. A esta función lineal, $z(W, b)$, le sigue una función de activación, $a(z)$, que da como resultado la salida, es decir, la activación de las neuronas de la siguiente capa. Este proceso lo realizan todas las neuronas en todas las capas hasta la capa de salida, que es el resultado de la red. En resumen, las neuronas aplican las siguientes funciones:

$$z_i^k = \sum_{j=1}^{n^{k-1}} (w_{i,j}^k * a_j^{k-1}) + b_i^k \tag{9}$$

$$a_i^k(z) = g(z_i^k) \tag{10}$$

Donde a_i^z es la salida de la neurona i de la capa k y $n^{(k-1)}$ es el número de neuronas

en la capa anterior. $w_{i,j}^k$ es el peso de la conexión entre la neurona j en la capa $k-1$ y la neurona i en la capa k, $g(x)$ es la función de activación y b_i^k es el sesgo de la neurona i en la capa k.

Una función de activación es una ecuación matemática que, a partir de la suma ponderada de las entradas, determina la salida de una neurona, (dicta si la neurona puede activarse o no). Las ANN se basan en funciones de activación que ayudan a aprender patrones complejos en los datos a través del proceso de retropropagación [17]. Algunas de las funciones de activación son:

1. **Función identidad**

$$a(x) = x.$$

2. **Función sigmoid o logística**

$$a(x) = \frac{1}{1 + exp(-x)}.$$

3. **Función tangente hiperbólica o *tanh***

$$a(x) = \frac{exp(x) - exp(-x)}{exp(x) + exp(-x)} = \frac{exp(2x) - 1}{exp(2x) + 1}.$$

4. **Función de unidad lineal rectificada o *ReLU***

$$a(x) = max(0, x).$$

Cuando se realiza el proceso directo para generar la predicción inicial, hay una función de error (E) o función de pérdida. Esta define qué tan lejos está el resultado del valor real. Hay dos funciones de pérdida de uso común:

$$E(y, \hat{y}) = -(y * log(\hat{y}) + (1 - y) * log(1 - \hat{y})) \tag{11}$$

$$E(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{12}$$

Donde m es el número total de muestras, y es el objetivo (dato medido) y $\hat{y}$ el valor predicho. Cada función de perdida sirve para un tipo particular de problema. La Ecuación (11) se denomina Cross Entropy, se usa en problemas de clasificación binaria. Por su parte, la Ecuación (12) es llamada Mean Squared Error, se utiliza para tareas de regresión.

El objetivo del entrenamiento de la red es encontrar un conjunto de pesos que minimice el valor de E para un conjunto de datos (datos de entrenamiento) La media de la función de pérdida aplicada a todas las muestras de la prueba de entrenamiento se conoce como función de costo. La influencia del peso individual en la función de pérdida se determina mediante retropropagación.

Para lograr el conjunto óptimo de pesos, se realiza el paso hacia atrás, retrocediendo desde la predicción de la red hasta las neuronas que generaron esa predicción. En pocas palabras, la retropropagación es un método para calcular la primera derivada de la función de error con respecto a cada peso de la red con el fin de encontrar pesos que minimicen la función de pérdida [21]. Se realiza mediante el algoritmo de descenso de gradiente (GD). Como la función MSE es una función convexa, es posible encontrar el óptimo global con GD, actualizando iterativamente los parámetros de la red (w, b).

El algoritmo de retropropagación puede estar compuesto por los siguientes pasos:

1. Cálculo del avance.
2. Retropropagación a la capa de salida.
3. Retropropagación a las capas ocultas.
4. Actualizaciones de parámetros

El algoritmo se detiene cuando el valor de la función de error se ha vuelto lo suficientemente pequeño. La actualización de los parámetros se realiza de la siguiente manera:

$$w_{i,j}^{k} = w_{i,j}^{k} - \alpha * \frac{\partial E}{\partial w_{i,j}^{k}} \tag{13}$$

$$b_{i}^{k} = b_{i}^{k} - \alpha * \frac{\partial E}{\partial b_{i}^{k}} \tag{14}$$

Donde α es un hiperparámetro de la red conocido como tasa de aprendizaje, que determina qué tan rápido se actualizan los parámetros. Si el aprendizaje es demasiado grande, se sobrepasarán los valores óptimos. Por el contrario, si es demasiado pequeño, la convergencia requerirá demasiadas iteraciones.

Con el GD clásico, los parámetros se actualizan después de que todo el conjunto de entrenamiento haya pasado por la propagación hacia adelante y hacia atrás, pero esto no suele ser práctico cuando se trabaja con grandes conjuntos de datos. Por lo general, se divide los datos en lotes [13]. El tamaño de lotes es un hiperparámetro que debe ajustarse para aumentar la eficiencia del proceso de entrenamiento. Además, una época en el aprendizaje automático significa un paso completo del conjunto de datos de entrenamiento a través del algoritmo. El número de épocas es un hiperparámetro importante para el aprendizaje del modelo, ya que con cada época, los parámetros del modelo se actualizan de acuerdo a los datos de entrenamiento permitiendo obtener nuevas características.

3 | Planteamiento del problema

A día de hoy, una de las técnicas de control que muestra mejor desempeño ante una variedad de problemas es el MPC. Sin embargo, esta técnica es computacionalmente costosa puesto que requiere solucionar una secuencia de problemas de optimización. Para dar solución a este problema, se propone estudiar modelos de inteligencia artificial que nos proporcionen un desempeño cercano al de un controlador MPC, con la diferencia de requerir menos recursos computacionales.Esto debido a que los modelos de IA únicamente necesitan evaluar una cierta función (obtenida en el proceso de entrenamiento) para proporcionar la acción de control requerida. Es decir, se suprime la necesidad de resolver iterativamente un problema de optimización.

Dado que nuestro diseño es exploratorio, hemos decidido emplear una planta simple con el fin de mostrar los resultados obtenidos en cada uno de los experimentos. La planta seleccionada consiste en un salón con una temperatura inicial T_0, una temperatura ambiente T_a y una temperatura objetivo T_r. Cabe mencionar que tanto T_0 como T_r son parámetros del sistema que pueden elegirse arbitrariamente.

El modelo de la planta con el que experimentaremos es el siguiente:

$$\frac{dT}{dt} = (T_a(t) - T(t)) + c(t) \tag{1}$$

$$\frac{dc}{dt} = q(t) \tag{2}$$

Donde $q(t)$ es la entrada de control (cuyo valor deberá decidir el algoritmo de inteligencia artificial). La variable $q(t)$ puede entenderse como la acción de subir o bajar la potencia de calefacción. Si $q(t)$ es positiva, entonces la potencia de calefacción sube con una velocidad proporcional al valor de $q(t)$. Si $q(t)$ es negativa, entonces la potencia de calefacción baja a una velocidad proporcional al valor de $q(t)$. La variable $c(t)$ es la potencia de calefacción aplicada al salón en el tiempo t y $T(t)$ es la temperatura del salón (que deseamos que esté cerca a T_r) en el tiempo t.

Seguidamente, diseñaremos un controlador MPC con el fin de generar los datos que permiten realizar el entrenamiento de las técnicas de IA, como por ejemplo, regresión lineal, vectores de soporte o redes neuronales. Así, obtenemos la información necesaria para realizar comparaciones entre el controlador MPC y los modelos de inteligencia artificial, ya que esta comparación permite mostrar si los modelos logran obtener resultados similares a un controlador MPC.

Una vez definida la planta, procedemos a diseñar el controlador MPC, el cual nos dará los valores apropiados para $q(t)$. Se decide el uso de MPC debido a su importancia en la industria actual, ya que es frecuente ver sus aplicaciones en diferentes escenarios obteniendo buenos resultados en cada uno de ellos. Finalmente seleccionamos modelos de regresión lineal, vectores de soporte y redes neuronales como modelos de inteligencia artificial para probar nuestras hipótesis. La selección de dichos modelos se debió a que son métodos tradicionales en el campo del ML.

4 | Metodología

Nuestra metodología se divide en 4 etapas, está orientada a resolver los siguientes problemas:

1. Seleccionar y modelar una planta sobre la que se hará la aplicación del controlador MPC.
2. Integrar inteligencia artificial para afrontar las limitaciones computacionales que tiene el MPC en la planta seleccionada.

A continuación describimos cada etapa:

1. **Implementación de la planta a controlar en un software de simulación**: en Matlab se implementó la planta descrita en la sección anterior. Para ello es necesario definir los valores de los parámetros que intervienen en el modelo;

   a) T_0 tomar el valor de 15.

   b) T_a toma los valores en el rango de 25 a 40.

   c) T_r toma los valores en el rango de 20 a 40.

   Para ver la implementación realizada, dirigirse al repositorio de Github del proyecto.

2. **Diseño del controlador MPC:** la síntesis del controlador MPC se hizo mediante prueba y error, buscando que el tiempo de establecimiento sea el menor posible y se minimice el sobrepaso. Las constantes de MPC elegidas fueron:

   a) Tiempo de muestreo $T_s = 0{,}5$ segundos.

b) Horizonte de predicción $H = 120$ segundos.

c) Horizonte de control $H_c = 120$ segundos.

Las simulaciones, bajo estos parámetros, arrojan la siguiente información, la cual se usa en la etapa 3:

a) Tiempo

b) Temperatura de la planta

c) Potencia de la calefacción

d) Temperatura ambiente

e) Temperatura objetivo

Por otro lado, encontramos los valores de respuesta del controlador, los cuales restrigen en un intervalo de entre $-0{,}5$ y $0{,}5$. A continuación se muestra una simulación de nuestro controlador MPC utilizando la estructura (4.1). En esta simulación se usan los siguientes parámetros $T_0 = 25$, $T_a = 25$ y $T_r = 20$.

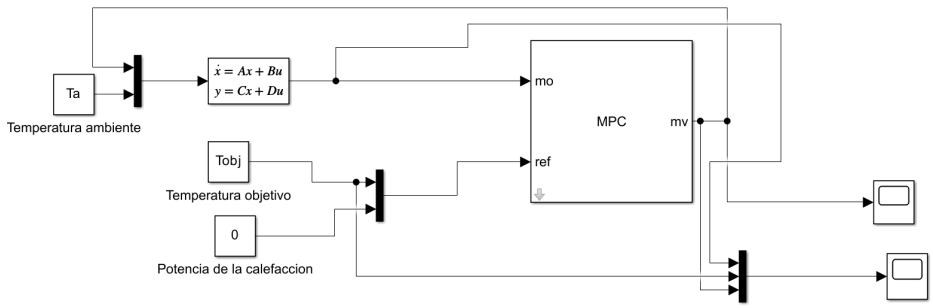

Figura 4.1: Planta modelada con el controlador MPC en matlab.

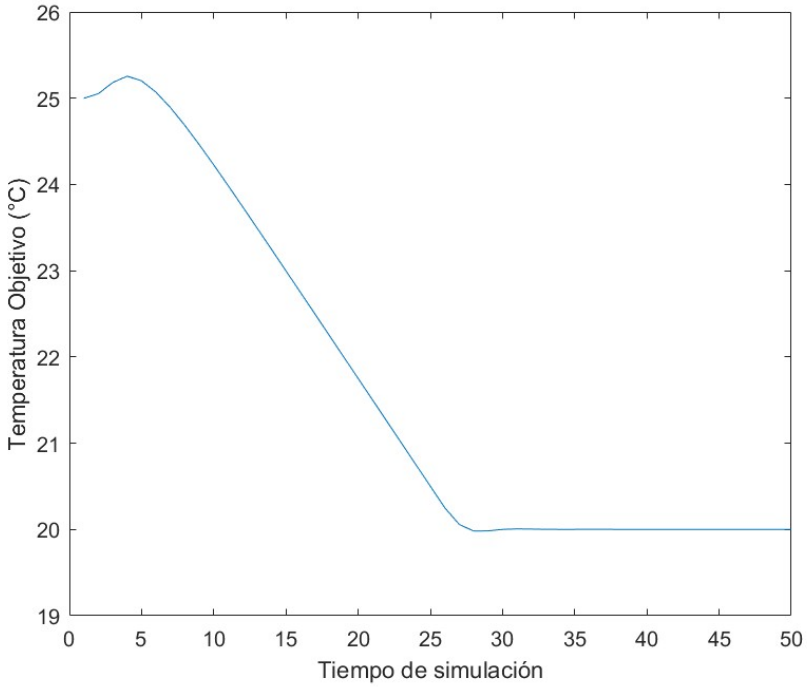

Figura 4.2: Resultado de la simulación utilizando el controlador MPC

En la figura 4.2 podemos observar cómo el controlador MPC logra alcanzar la referencia de temperatura establecida sin error en estado estacionario y con un sobrepaso mínimo.

3. **Entrenamiento de los modelos de inteligencia artificial :** para entrenar los modelos de inteligencia artificial utilizamos los datos generados a través del modelo implementados en Matlab. A continuación procedemos a realizar una descripción sobre las características presentes en cada modelo, cabe mencionar que los datos se separan en datos de entrenamiento y datos de validación para verificar el correcto entrenamiento de cada uno de los modelos.

   a) **Regresión lineal:** para la implementación de la regresión lineal utilizamos la librería *sklearn*, en donde los parámetros seleccionados son los configurados por defecto en la librería. El entrenamiento arroja la siguiente ecuación $y = -0{,}054T - 0{,}003T_a + 0{,}056T_r + 0{,}034$, donde y representa la acción de control.

   b) **Maquina de soporte vectorial:** para realizar el entrenamiento del algo-

ritmo de máquina de soporte vectorial es necesario realizar antes una estandarización de los datos, debido a la sensibilidad del modelo. Luego procedemos a realizar el entrenamiento del modelo a través de la librería de *sklearn*, cuyos hiperparámetros son los definidos por defecto. Obtuvimos la siguientes ecuación $y = -2{,}84 \times 10^{-1}T + 1{,}78 \times 10^{-5}T_a + 2{,}80 \times 10^{-1}T_r - 4{,}51 \times 10^{-3}$.

c) **Red neuronal:** para la construcción de la red neuronal utilizamos tres bloques, los cuales describimos a continuación. El primer bloque es una capa densa de 128 neuronas y un kernel normal. Luego, la salida de dicha capa se inserta a una nueva capa de normalización con una función de activación lineal. El segundo bloque esta compuesto por 4 capas ocultas con kernels normales. El número de neuronas de cada capa oculta es $256, 256, 128$ y 32 respectivamente. Por último, la tercera capa está formada por un kernel normal y una neurona, a continuación podemos observar a grande rasgos la estructura de la red neuronal.

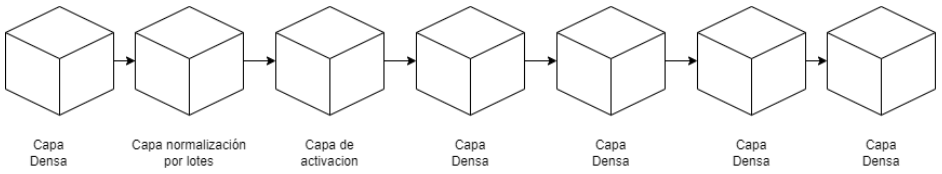

Figura 4.3: Arquitectura de la red neuronal.

Para el entrenamiento de nuestra red neuronal, primero se estandarizan los datos. Como función de pérdida se elige el *mean absolute error*. El algoritmo de optimización que se utiliza es *adam* y como métricas, el *mean absolute error*. Finalmente, entrenamos con 8 épocas y obtenemos la siguiente gráfica donde se puede observar una variación sobre el aprendizaje del error.

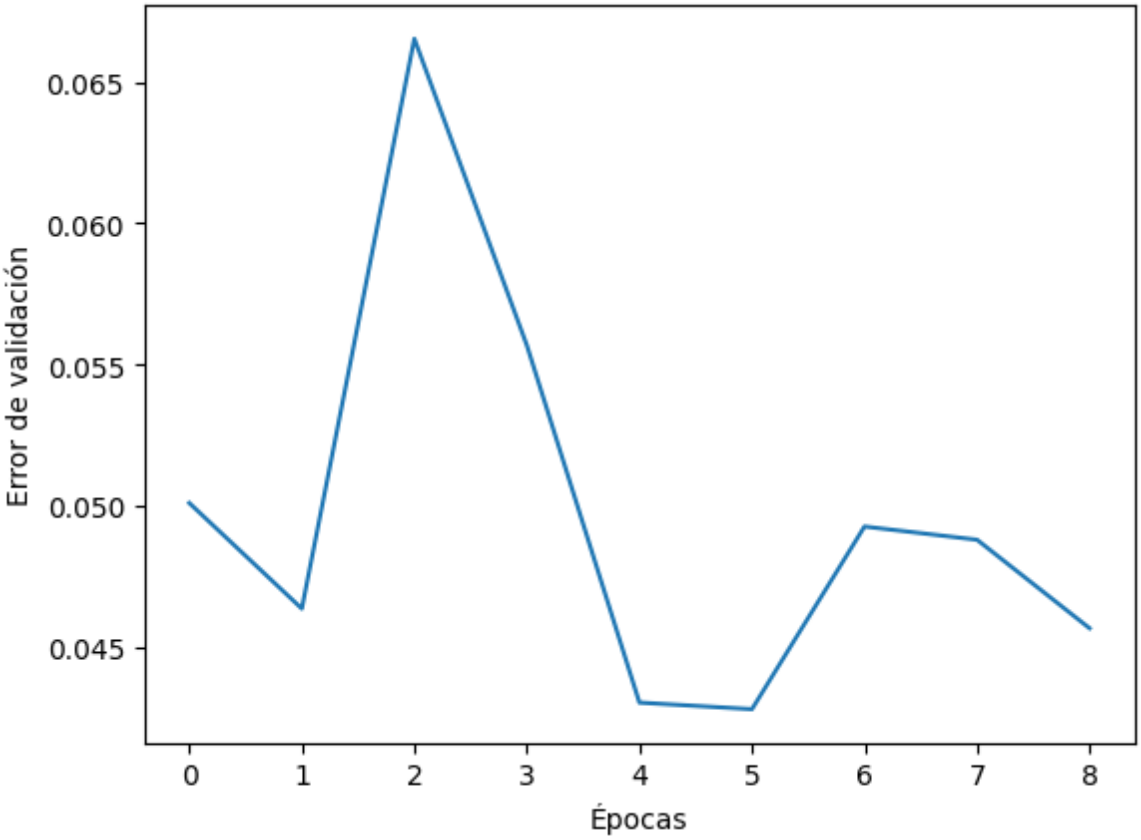

Figura 4.4: Función de perdida a través de las épocas

4. **Resultados:** en la etapa de resultados, consiste en la comparación del desempeño de los modelos de IA respecto al controlador MPC. Los resultados de esta etapa se describen detalladamente en la siguiente sección.

5 | Resultados

Esta sección incluye el análisis de los resultados obtenidos a partir de los modelos de inteligencia artificial que se han descrito anteriormente en este documento. Cabe mencionar que los resultados tienen como base un conjunto de datos de 1200 elementos que se genera a través del modelo (2), su característica se encuentra en el cambio de la variable T_0, T_r, donde toman valores entre 20 y 25, mientras que T_a toma el valor fijo de 25. De esta forma, obtenemos los datos necesarios para evaluar los modelos.

El primer resultado, que se muestra en la Figura (5.1), se muestra la comparación de la señal de control obtenida con MPC y cada uno de los modelos de ML. Podemos ver que los modelos de ML se acercan al comportamiento del controlador MPC. De hecho, el modelo que más se acerca es el de la red neuronal. Sin embargo, hay que resaltar que el MPC usa todo el rango de control disponible (es decir, las señales de control generadas usando MPC llegan a los límites del intervalo permitido $[-0{,}5, 0{,}5]$).

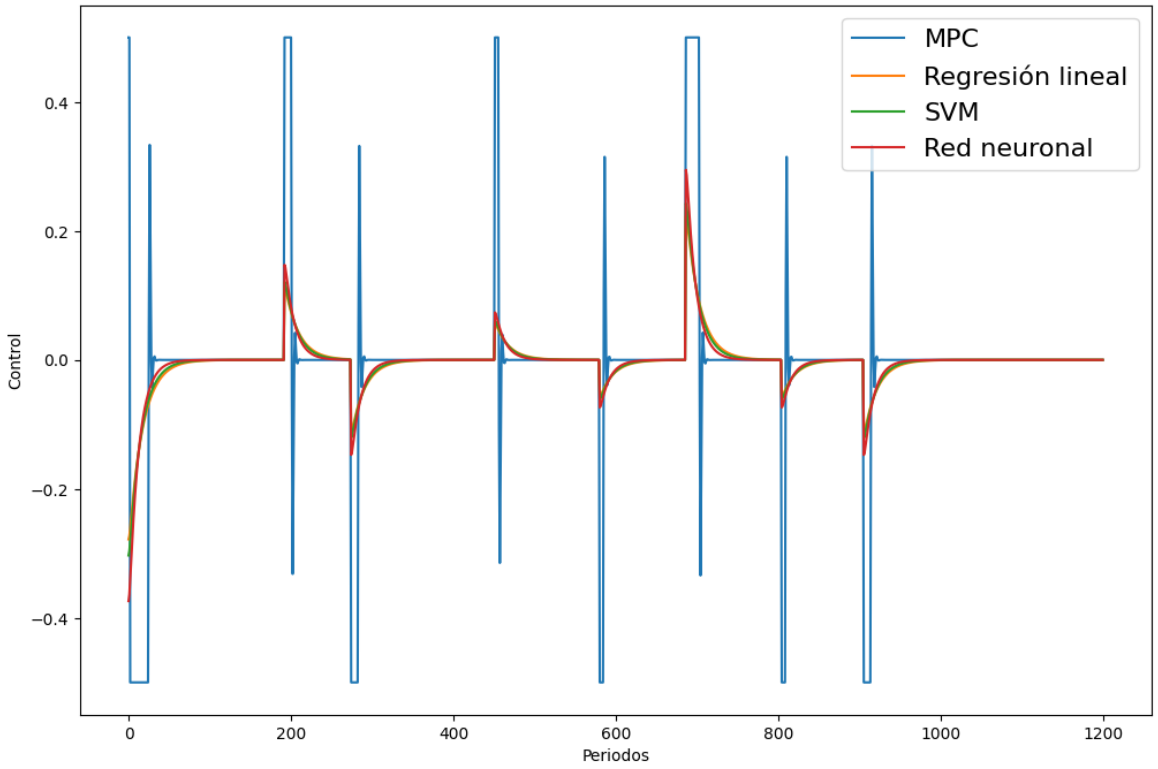

Figura 5.1: Respuesta de control vs modelos ML

Nuestro segundo resultado, que se presenta en la Figura (5.2), muestra el seguimiento de la referencia de temperatura (temperatura deseada) bajo cada una de las estrategias de control. Nuevamente, el controlador MPC es el que mejor desempeño tiene, puesto que el tiempo de establecimiento (tiempo en llegar a la referencia) es significativamente más pequeño comparado con los modelos de ML. Sin embargo, cabe resaltar que los controladores basados en ML son capaces de hacer un seguimiento de la referencia. De hecho, el único método que presenta un error de estado estable relativamente alto es el de regresión lineal.

Algo interesante a destacar, es que según el Cuadro (5.1), en el que se muestran algunas métricas de entrenamiento de cada uno de los modelos de ML usados (específicamente, se muestra el error cuadrático medio (MSE) y el error cuadrado (R^2)), la regresión lineal es la que mejor desempeño presenta. Sin embargo, en la Figura (5.2) se puede observar que la regresión lineal no tiene un buen comportamiento a la hora de controlar la planta. Esto puede deberse a que las métricas estándares

usadas en ML evalúan el ajuste a los datos sin considerar el desempeño dinámico del algoritmo cuando es usado como un controlador. Por este motivo, decidimos comparar el comportamiento de cada uno de los modelos a la luz de una de las métricas de control más usadas: el índice ISE (integral del error al cuadrado). Este índice, básicamente mide qué tan desviada está la variable a controlar de su valor de referencia a lo largo de un tiempo determinado (que en nuestro caso es el tiempo de simulación). La fórmula para calcular el índice ISE es la siguiente:

$$ISE = \int_0^T (r(t) - y(t))^2 dt$$

En esta fórmula, $r(t)$ es el valor de referencia de la variable a controlar en el tiempo t, $y(t)$ es el valor real de la variable a controlar en el tiempo t, y T es el tiempo total de la simulación. La integral se realiza sobre el intervalo $[0, T]$. La idea detrás de esta fórmula es simple: se calcula el error cuadrático entre la variable a controlar y su valor de referencia en cada instante de tiempo, y luego se suma ese error cuadrático a lo largo del tiempo. La integral se usa para sumar los errores acumulados a lo largo del tiempo, produciendo una métrica que mide el desempeño del controlador en todo el período de tiempo. Por lo tanto, comparar el comportamiento de los modelos usando el índice ISE permite evaluar no solo el ajuste a los datos, sino también el desempeño dinámico del controlador a lo largo del tiempo.

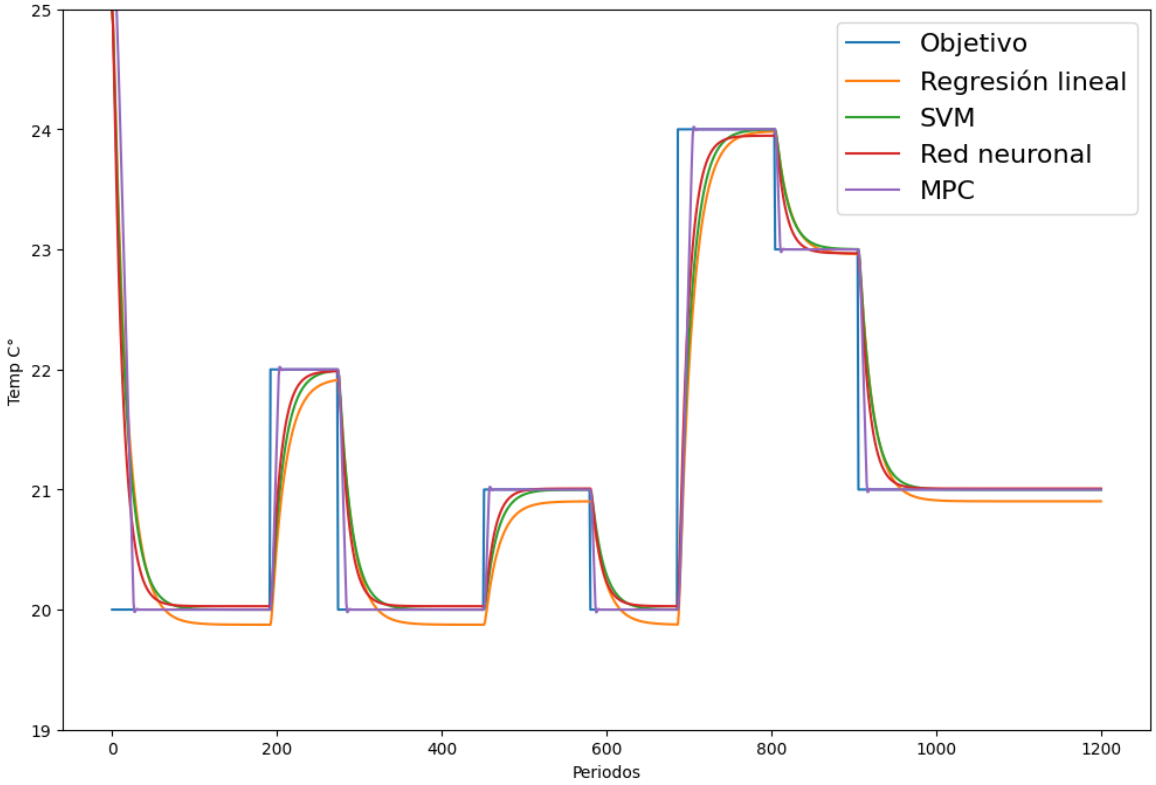

Figura 5.2: Temperatura objetivo vs modelos ML

Controlador	MSE	R^2
Regresión Lineal	0.017	0.593
Vectores de soporte	0.017	0.581
Red neuronal	0.025	0.407

Cuadro 5.1: Error MSE y R^2

Controlador	ISE
MPC	95.717
Red Neuronal	120.665
SVM	146.378
Regresión lineal	171.827

Cuadro 5.2: Integral del error cuadrático

En el Cuadro (5.2) se puede notar que la red neuronal es la que mejor desempeño tiene entre los modelos de ML. Aun así, hay que observar que MPC sigue liderando en este ítem. Por su parte, la regresión lineal es el modelo que muestra el menor rendimiento.

Por último, diseñamos un experimento con los modelos ya entrenados para determinar el tiempo computacional de cada uno de ellos, hay que recordar que nuestra hipótesis inicial; los modelos de ML son capaces de emular el comportamiento de un MPC con un costo computacional menor. Para ello se consideraron los siguientes parámetros:

1. T_a igual a 25.
2. T_0 igual a 25.
3. T_r conjunto de valores entre 10 y 34.

Se obtiene el siguiente resultado:

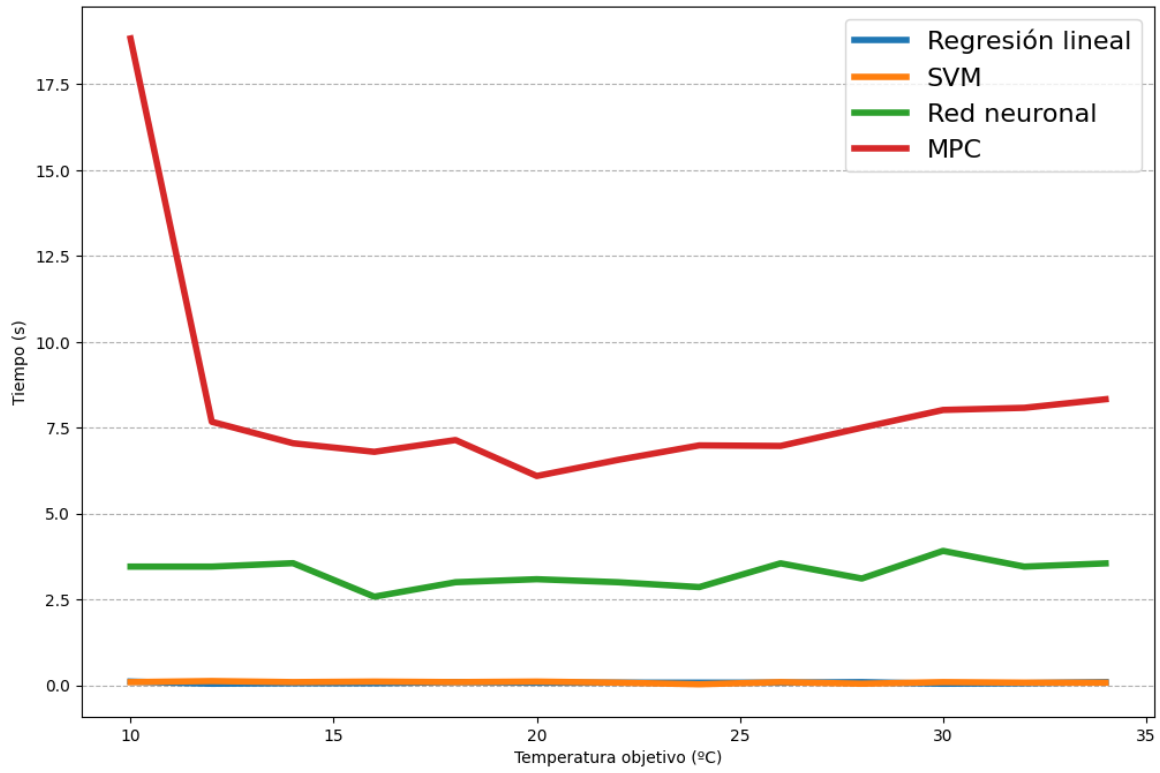

Figura 5.3: Tiempo computacional de cada controlador

Como se puede ver el Figura (5.3), el tiempo empleado por los modelos de ML es menor a MPC, lo cual valida nuestra hipótesis inicial, mostrando la mejora computacional que trae utilizar ML para emular controladores.

6 | Conclusiones

Finalmente, se puede afirmar que los objetivos planteados en esta tesis se han logrado desarrollar con éxito, ya que como se pudo ver, se diseñó una planta con el controlador MPC con éxito y además se implementaron modelos de ML que logran emular el comportamiento de un controlador MPC.

El tiempo de ejecución, como se puede ver en la Figura (5.3), se logra mejorar con la ayuda de los modelos de ML, logrando así cumplir con el objetivo de reducir el costo computacional de un controlador MPC convencional.

Por otro lado, se deben resaltar los resultados obtenidos con las métricas tradicionales de ML. La forma estándar de extraer estas métricas, se basa en tomar punto a punto cada uno de los valores resultantes de las pruebas y calcular su error, Sin embargo, como se observó en este trabajo, es necesario emplear métricas distintas a las tradicionales cuando se está evaluando el desempeño de algoritmos de ML aplicados a problemas dinámicos, como es el caso del control de sistemas.

Para futuros proyectos, un posible escenario de estudio sería aplicar estos modelos de ML a algún sector de la industria para así poder evaluar el comportamiento a gran escala de los controladores basados en ML. Además, sería interesante estudiar el comportamiento de las redes neuronales recurrentes (RNN) en el mismo sistema, ya que estos modelos tienen la capacidad de recordar información del pasado, lo que podría resultar útil en aplicaciones de control.

Bibliografía

[1] A. Bensoussan. *Estimation and control of dynamical systems*, volume 48. Springer, 2018.

[2] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[3] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer science & business media, 2013.

[4] N. Castle. Supervised vs. unsupervised machine learning. *Retrieved from*, 2017.

[5] N. Castle. What is semi-supervised learning. *Oracle DataScience. Com*, 2018.

[6] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[7] B. Chang, L. Meng, E. Haber, F. Tung, and D. Begert. Multi-level residual networks from dynamical systems view. *arXiv preprint arXiv:1710.10348*, 2017.

[8] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[9] F. J. Fernández-Ovies, E. Santiago Alférez-Baquero, E. J. de Andrés-Galiana, A. Cernea, Z. Fernández-Muñiz, and J. L. Fernández-Martínez. Detection of

breast cancer using infrared thermography and deep neural networks. In *Bioinformatics and Biomedical Engineering: 7th International Work-Conference, IWBBIO 2019, Granada, Spain, May 8-10, 2019, Proceedings, Part II 7*, pages 514–523. Springer, 2019.

[10] H. J. Ferreau, S. Almér, H. Peyrl, J. L. Jerez, and A. Domahidi. Survey of industrial applications of embedded model predictive control. In *2016 European Control Conference (ECC)*, pages 601–601. IEEE, 2016.

[11] J. Jobson and J. Jobson. Multiple linear regression. *Applied multivariate data analysis: Regression and experimental design*, pages 219–398, 1991.

[12] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[13] W. Koehrsen. Overfitting vs. underfitting: A complete example. *Towards Data Science*, 2018.

[14] F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[15] C. Ma, L. Wu, et al. Machine learning from a continuous viewpoint, i. *Science China Mathematics*, 63(11):2233–2266, 2020.

[16] B. Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386, 2020.

[17] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[18] S. Prajna, P. A. Parrilo, and A. Rantzer. Nonlinear control synthesis by convex optimization. *IEEE Transactions on Automatic Control*, 49(2):310–314, 2004.

[19] R. Raj. *Java Deep Learning Cookbook: Train neural networks for classification, NLP, and reinforcement learning using Deeplearning4j*. Packt Publishing Ltd, 2019.

[20] B. Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019.

[21] R. Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[22] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *arXiv preprint arXiv:1704.02532*, 2017.

[23] P. O. Scokaert, D. Q. Mayne, and J. B. Rawlings. Suboptimal model predictive control (feasibility implies stability). *IEEE Transactions on Automatic Control*, 44(3):648–654, 1999.

[24] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[25] V. Vapnik. *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

[26] K. W. Wong, G. Franciolini, V. De Luca, V. Baibhav, E. Berti, P. Pani, and A. Riotto. Constraining the primordial black hole scenario with bayesian inference and machine learning: the gwtc-2 gravitational wave catalog. *Physical Review D*, 103(2):023026, 2021.

[27] B. Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[28] G. Zhang, B. E. Patuwo, and M. Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.