DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA ELECTRÓNICA EN AUTOMATIZACIÓN Y CONTROL

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ELECTRÓNICO EN AUTOMATIZACIÓN Y CONTROL

TEMA: DESARROLLO DE CONTROLADORES CON REDES NEURONALES DE APRENDIZAJE PROFUNDO APLICANDO TENSORFLOW

AUTOR: SÁNCHEZ ALBÁN, PABLO ISRAEL

DIRECTOR: ING. PROAÑO ROSERO, VÍCTOR GONZALO

SANGOLQUÍ

2020

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA ELECTRÓNICA EN AUTOMATIZACIÓN Y CONTROL

CERTIFICACIÓN

Certifico que el trabajo de titulación, "**DESARROLLO DE CONTROLADORES CON REDES NEURONALES DE APRENDIZAJE PROFUNDO APLICANDO TENSORFLOW**" realizado por el señor ***Sánchez Albán, Pablo Israel*** el mismo que ha sido revisado en su totalidad, analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente

Sangolquí, 20 de diciembre de 2019

Firma

.....

Ing. Víctor Gonzalo Proaño Rosero

C.C. 1706457924

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA ELECTRÓNICA EN AUTOMATIZACIÓN Y CONTROL

AUTORÍA DE RESPONSABILIDAD

Yo, ***Sánchez Albán, Pablo Israel***, con cédula de identidad N° 1724698129, declaro que este trabajo de titulación: "**DESARROLLO DE CONTROLADORES CON REDES NEURONALES DE APRENDIZAJE PROFUNDO APLICANDO TENSORFLOW**" es de mi autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Consecuentemente el contenido de la investigación mencionada es veraz.

Sangolquí, 20 de diciembre de 2019

Sánchez Albán Pablo Israel

C.C. 1724698129

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA ELECTRÓNICA EN AUTOMATIZACIÓN Y CONTROL

AUTORIZACION

Yo, ***Sánchez Albán, Pablo Israel*** autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **DESARROLLO DE CONTROLADORES CON REDES NEURONALES DE APRENDIZAJE PROFUNDO APLICANDO TENSORFLOW** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad

Sangolquí, 20 de diciembre de 2019

Sánchez Albán Pablo Israel

C.C. 1724698129

DEDICATORIA

Este trabajo va dedicado a mi Dios Eterno, que con gracia y amor me ha guiado hasta este punto de mi vida.

A mis padres, hermanos y a mi cuñada Renate, que me han brindado su apoyo y cariño incondicional.

A mis amigos Reno, Henry y a mis amigos de la ESPE, con los que he vivido momentos inolvidables y me han ayudado a ser mejor persona.

Pablo

AGRADECIMIENTOS

Mi agradecimiento más profundo a Dios, León de la tribu de Judá, por nunca dejarme, por darme fuerzas y luchar a mi lado en todo momento. Que lo yo he logrado sea para honra y gloria de Él.

A mis padres, hermanos y a mi cuñada Renate por sus palabras, por siempre esperar lo mejor de mí, por confiar y cuidarme. Gracias a ellos me he sentido motivado y con ganas de seguir adelante.

Quedo profundamente agradecido con el ingeniero Víctor Proaño, por su guía en la realización de este trabajo. Gracias a él este trabajo ha sido desarrollado de la mejor manera alcanzando la excelencia.

Pablo

INDICE DE CONTENIDOS

CERTIFICACIÓN i
AUTORÍA DE RESPONSABILIDAD ii
AUTORIZACIÓN iii
DEDICATORIA iv
AGRADECIMIENTOS v
INDICE DE CONTENIDOS vi
INDICE DE TABLAS ix
INDICE DE FIGURAS x
RESUMEN xv
ABSTRACT xvi
CAPÍTULO I 1
1. INTRODUCCION 1
1.1 Antecedentes 1
1.2 Justificación 3
1.3 Alcance del proyecto 4
1.4 Objetivos 5
1.4.1 Objetivo General 5
1.4.2 Objetivos Específicos 5
CAPÍTULO II 6
2. MARCO TEÓRICO 6
2.1. Redes Neuronales 6
2.1.1. Neurona biológica 6
2.1.2. Neurona artificial 7
2.1.3. Redes neuronales de aprendizaje profundo 8
2.1.4. Estructura de una red neuronal 8
2.1.5. Tipos de entrenamiento 9
2.2. Descripción de la librería Tensorflow 11
2.2.1. Características de Tensorflow 12
2.2.2. Red neuronal en TensorFlow 12
2.2.3. Tipos de modelos Redes Neuronales 13
2.2.4. Funciones principales 16
2.3. Herramienta gráfica 23

2.4. Componentes de TensorFlow 24
2.4.1. Tensor 25
2.4.2. Nodos 25
2.4.3. Grafo 25
2.4.4. Sesión 26
2.4.5. Dispositivos 26
2.5. Sistemas Dinámicos 26
2.5.1. Identificación de Sistemas Dinámicos 27
2.5.2. Control Neuronal Inverso 29
2.5.3. Control Neuronal con modelo de referencia 30
2.6. Instalación de software necesario. 30
2.6.1. Anaconda Python Distribution 30
CAPÍTULO III 36
3. DESARROLLO DE ALGORITMOS BASICOS Y CONTROL REALIMENTADO 36
3.1. Ejemplos de Aplicaciones Básicas 36
3.1.1. Problema XOR 36
3.1.2. Identificación de funciones 39
3.2. Ejemplos de Aplicación de Identificación 43
3.2.1. Identificación del sistema Antena con péndulo invertido 43
3.2.2. Identificación del sistema Tanques Acoplados 50
3.2.3. Identificación del sistema Viga y Bola 57
3.3. Ejemplos de aplicación de control con red neuronal inversa 65
3.3.1. Control con red neuronal inversa Antena con péndulo invertido 65
3.3.2. Control con red neuronal inversa Tanques Acoplados 69
3.3.3. Control con red neuronal inversa Viga y Bola 73
3.4. Ejemplos de aplicación de control por Modelo de Referencia 79
3.4.1. Control con modelo de referencia para el sistema Antena 79
3.4.2. Control con modelo de referencia para el sistema Tanques Acoplados 87
3.4.3. Control con modelo de referencia para el sistema Viga y Bola 95
CAPÍTULO IV 104
4. CONCLUSIONES Y RECOMENDACIONES 104
4.1. Conclusiones 104

4.2. Recomendaciones 105
CAPÍTULO V 106
5. BIBLIOGRAFIA 106

INDICE DE TABLAS

Tabla 1 *Lista de funciones de activación* 17
Tabla 2 *Lista de optimizadores, funciones de coste y métricas para la función 'compile'*.....20
Tabla 3 *Tabla de la Función XOR*.....36
Tabla 4 *Comparación de salida deseada y estimada de red neuronal XOR*.....38

INDICE DE FIGURAS

Figura 1. Esquema de MRL 4
Figura 2. Neurona biológica y sus componentes 6
Figura 3. Modelo de una neurona artificial 7
Figura 4. Estructura de una red neuronal 8
Figura 5. Aprendizaje supervisado 10
Figura 6. Aprendizaje no supervisado 10
Figura 7. Secuencia de entrenamiento de red neuronal 12
Figura 8. Importación del conjunto de patrones de entrenamiento 13
Figura 9. Ejemplo de estructura de Modelo Secuencial 14
Figura 10. Ejemplo de estructura de Modelo Funcional 15
Figura 11. Sintaxis de función Dense para modelo secuencial 16
Figura 12. Sintaxis de función Dense para modelo funcional 16
Figura 13. Sintaxis de función Compile 19
Figura 14. *Sintaxis de función 'fit'* 22
Figura 15. *Sintaxis de función 'predictt'* 23
Figura 16. *Sintaxis de función 'summary'* 23
Figura 17. *Código para exportar el modelo a una imagen png* 24
Figura 18. Componentes de TensorFlow 24
Figura 19. Esquema de un grafo en TensorFlow 26
Figura 20. Representación de modelo dinámico 27
Figura 21. Obtención de variaciones de estado 27
Figura 22. Diagrama identificación 28
Figura 23. Señal de salida de red neuronal con retardo de primer orden 28
Figura 24. Diagrama controlador neuronal inverso 29
Figura 25. Esquema de control con red neuronal inversa 29
Figura 26. Esquema de control con red neuronal con modelo de referencia 30
Figura 27. Página de descarga de Anaconda 31
Figura 28. Interfaz gráfica de Anaconda Navigator 32
Figura 29. Creación de un nuevo Ambiente 33
Figura 30. Instalación de Jupyter Notebook 33
Figura 31. Anaconda Prompt 34
Figura 32. Activación del Ambiente en consola Anaconda Prompt. 34
Figura 33. Comandos para instalar librerías necesarias. 34
Figura 34. Intefaz de Jupyter Notebook 35
Figura 35. Importación librerías – Identificación Antena 35
Figura 36. DLTI for TensorFlow-Keras Models 35
Figura 37. Patrones de entrenamiento para XOR 37
Figura 38. Creación del modelo XOR 37
Figura 39. Resumen red neuronal XOR 37
Figura 40. Compilación de red neuronal XOR 38
Figura 41. Entrenamiento de red neuronal XOR 38

Figura 42. Predicción de la red neuronal XOR usando todas las combinaciones posibles. 38
Figura 43. Resultados RN XOR 39
Figura 44. Patrones de entrenamiento – Identificación de funciones 39
Figura 45. Creación del modelo – Identificación de funciones 40
Figura 46. Resumen red neuronal – Identificación de funciones 40
Figura 47. Compilación de red neuronal – Identificación de funciones 40
Figura 48. Entrenamiento de red neuronal – Identificación de funciones 41
Figura 49. Grafica de la función $z = x2 + y2$ 41
Figura 50. Grafica de la función identificada $z = x2 + y2$ 42
Figura 51. Antena con péndulo invertido 43
Figura 52. Diagrama de bloques Antena con péndulo invertido 44
Figura 53. Diagrama de Antena para obtención de variación de variables de estado 45
Figura 54. Script para adquisición de patrones de entrenamiento - Identificación Antena 45
Figura 55. Patrones de entrenamiento - Identificación Antena 46
Figura 56. Creación del modelo en Python – identificación Antena 46
Figura 57. Resumen red neuronal - identificación Antena 46
Figura 58. Compilación de red neuronal identificación – Antena 47
Figura 59. Entrenamiento de red neuronal identificación - Antena 47
Figura 60. Prueba de rendimiento RN identificación - Antena 47
Figura 61. Exportación de red neuronal identificación - Antena 48
Figura 62. Función en MATLAB para uso del modelo importado – Identificación Antena 48
Figura 63. Modelo RN Antena 49
Figura 64. Sistema vs RN - Identificación Antena 49
Figura 65. Curvas de variables de estado en lazo abierto – Antena 50
Figura 66. Esquema Tanques Acoplados 50
Figura 67. Esquema en Simulink de 2 tanques acoplados. 51
Figura 68. Esquema en Simulink tanque superior. 51
Figura 69. Esquema en Simulink tanque inferior. 52
Figura 70. Diagrama de Tanque Acoplados para obtención de variación de variables de estado 52
Figura 71. Adquisición de patrones de entrenamiento – Identificación Tanques Acoplados 53
Figura 72. Patrones de entrenamiento identificación – Identificación Tanques Acoplados 53
Figura 73. Creación del modelo en Python – Identificación Tanques Acoplados 54
Figura 74. Resumen red neuronal - Identificación Tanques Acoplados 54
Figura 75. Compilación de red neuronal – Identificación Tanques Acoplados 54
Figura 76. Entrenamiento de red neuronal – Identificación Tanques Acoplados 54
Figura 77. Prueba de rendimiento RN – Identificación Tanques Acoplados 55
Figura 78. Exportación de red neuronal – Identificación Tanques Acoplados 55
Figura 79. Función en MATLAB – Identificación Tanques Acoplados 56
Figura 80. Modelo RN – Identificación Tanques Acoplados 56
Figura 81. Sistema vs RN – Identificación Tanques Acoplados 56
Figura 82. Curvas de variables de estado en lazo abierto – Identificación Tanques Acoplados 57
Figura 83. Diagrama de viga y bola 57
Figura 84. Diagrama de bloques - Viga y Bola 58

Figura 85. Diagrama para obtención de variación de variables de estado – Viga y Bola.....59
Figura 86. Script para adquisición de patrones de entrenamiento Identificación – Viga y Bola..60
Figura 87. Patrones de entrenamiento – Viga y Bola.....60
Figura 88. Creación del modelo en Python – Viga y Bola.....61
Figura 89. Resumen red neuronal identificación – Viga y Bola.....61
Figura 90. Compilación de red neuronal identificación – Viga y Bola 61
Figura 91. Entrenamiento de red neuronal identificación – Viga y Bola.....62
Figura 92. Prueba de rendimiento RN identificación – Viga y Bola 62
Figura 93. Exportación de red neuronal identificación – Viga y Bola.....62
Figura 94. Función en MATLAB– Identificación Viga y Bola 63
Figura 95. Modelo RN – Viga y Bola 63
Figura 96. Sistema vs RN Identificación – Viga y Bola 64
Figura 97. Curvas de variables de estado en lazo abierto – Viga y Bola.....64
Figura 98. Script para adquisición de patrones de entrenamiento – RN inversa Antena.....65
Figura 99. Patrones de entrenamiento en Python – RN inversa Antena 65
Figura 100. Creación del modelo – RN inversa Antena 66
Figura 101. Resumen red neuronal – RN inversa Antena.....66
Figura 102. Compilación de red neuronal – RN inversa Antena 67
Figura 103. Exportación de red neuronal – RN inversa Antena 67
Figura 104. Función en MATLAB para uso del modelo importado – RN inversa Antena 67
Figura 105. Modelo – RN inversa Antena 68
Figura 106. Sistema controlado por RN inversa – Antena.....68
Figura 107. Respuesta de Control RN inversa – Antena 69
Figura 108. Script para adquisición de patrones de entrenamiento – RN inversa Tanques.....70
Figura 109. Patrones de entrenamiento en Python – RN inversa Tanques 70
Figura 110. Creación del modelo – RN inversa Tanques 70
Figura 111. Resumen red neuronal – RN inversa Tanques.....71
Figura 112. Compilación de red neuronal – RN inversa Tanques 71
Figura 113. Exportación de red neuronal – RN inversa Tanques 71
Figura 114. Función en MATLAB para uso del modelo importado – RN inversa Tanques 72
Figura 115. Modelo – RN inversa Tanques 72
Figura 116. Sistema controlado por RN inversa – Tanques.....72
Figura 117. Respuesta de Control RN inversa - Tanques 73
Figura 118. Script para adquisición de patrones de entrenamiento – RN inversa Viga.....74
Figura 119. Patrones de entrenamiento en Python – RN inversa Viga y Bola 74
Figura 120. Creación del modelo – RN inversa Viga y Bola.....74
Figura 121. Resumen red neuronal – RN inversa Viga y Bola.....75
Figura 122. Compilación de red neuronal – RN inversa Viga y Bola 75
Figura 123. Entrenamiento de red neuronal – RN inversa Viga y Bola 75
Figura 124. Prueba de rendimiento – RN inversa Viga y Bola.....76
Figura 125. Exportación de red neuronal – RN inversa Viga y Bola 76
Figura 126. Función en MATLAB para uso del modelo importado – RN inversa Viga y Bola ..77
Figura 127. Modelo – RN inversa Viga y Bola 77

Figura 128. Sistema controlado por RN inversa Viga y Bola.....78
Figura 129. Curvas de variables de estado en lazo abierto – RN inversa Viga y Bola.....78
Figura 130. Diagrama de bloques – MRL Antena.....79
Figura 131. Adquisición de patrones de entrenamiento – MRL Antena.....80
Figura 132. Patrones de entrenamiento – MRL Antena.....81
Figura 133. Importación Modelo Identificado – MRL Antena.....81
Figura 134. Creación del modelo en Python – MRL Antena.....82
Figura 135. Resumen red neuronal – MRL Antena.....83
Figura 136. Compilación de red neuronal identificación – MRL Antena.....83
Figura 137. Entrenamiento de red neuronal identificación – MRL Antena.....83
Figura 138. Extracción de Pesos y Bias de controlador – MRL Antena.....84
Figura 139. Creación de modelo para controlador – MRL Antena.....84
Figura 140. Exportación de red neuronal – MRL Antena.....84
Figura 141. Función en MATLAB – MRL Antena.....85
Figura 142. Modelo RN – MRL Antena.....85
Figura 143. Sistema controlado – MRL Antena.....86
Figura 144. Curvas de variables de estado sistema controlado – MRL Antena.....86
Figura 145. Sistema controlado por PID – MRL Tanques.....87
Figura 146. Adquisición de patrones de entrenamiento MATLAB – MRL Tanques.....88
Figura 147. Patrones de entrenamiento – MRL Tanques.....89
Figura 148. Entradas normalizadas – MRL Tanques.....89
Figura 149. Importación Modelo Identificado – MRL Tanques.....89
Figura 150. Creación del modelo en Python – MRL Tanques.....90
Figura 151. Resumen red neuronal – MRL Tanques.....91
Figura 152. Compilación de red neuronal identificación – MRL Tanques.....91
Figura 153. Entrenamiento de red neuronal identificación – MRL Tanques.....91
Figura 154. Extracción de Pesos y Bias de controlador – MRL Tanques.....92
Figura 155. Concatenación de pesos de la última capa del controlador – MRL Tanques.....92
Figura 156. Creación de modelo para controlador – MRL Tanques.....92
Figura 157. Exportación de red neuronal – MRL Tanques.....93
Figura 158. Función en MATLAB neuronal – MRL Tanques.....93
Figura 159. Modelo RN – MRL Tanques.....94
Figura 160. Sistema controlado – MRL Tanques.....94
Figura 161. Curvas de variables de estado sistema controlado – MRL Tanques.....95
Figura 162. Sistema controlado por PID – MRL Viga y Bola.....95
Figura 163. Adquisición de patrones de entrenamiento – MRL Viga y Bola.....96
Figura 164. Patrones de entrenamiento – MRL Viga y Bola.....97
Figura 165. Importación Modelo Identificado – MRL Viga y Bola.....98
Figura 166. Creación del modelo en Python – MRL Viga y Bola.....98
Figura 167. Resumen red neuronal – MRL Viga y Bola.....99
Figura 168. Compilación de red neuronal – MRL Viga y Bola.....99
Figura 169. Entrenamiento de red neuronal – MRL Viga y Bola.....99
Figura 170. Extracción de Pesos y Bias de controlador – MRL Viga y Bola.....100

Figura 171. Creación de modelo para controlador – MRL Viga y Bola.....100
Figura 172. Compilación modelo del controlador – MRL Viga y Bola 100
Figura 173. Exportación de red neuronal – MRL Viga y Bola.....101
Figura 174. Función en MATLAB – MRL Viga y Bola.....101
Figura 175. Modelo RN – MRL Viga y Bola 102
Figura 176. Sistema controlado – MRL Viga y Bola.....102
Figura 177. Curvas de variables de estado sistema controlado – MRL Viga y Bola.....103

RESUMEN

En el presente proyecto se desarrollaron algoritmos de redes neuronales usando la librería TensorFlow, para la solución de aplicaciones básicas como el problema XOR, el cual no es separable linealmente, la identificación de una función sencilla ($z = x^2 + y^2$). Por medio de las redes neuronales que se pueden entrenar usando esta herramienta también se realiza la identificación, control neuronal inverso y control neuronal con modelo de referencia lineal de sistemas dinámicos (Antena con péndulo invertido, Tanques Acoplados, Viga y Bola). Las redes neuronales fueron diseñadas y entrenadas en el lenguaje de Python y luego exportados a MATLAB R2017b. Las redes se han entrenado usando aprendizaje supervisado e implementando varias funciones de activación en cada una de sus capas, principalmente "ReLU". Para el desarrollo y para la trasferencia entre Python y MATLAB del modelo entrenado, se usa una API (Application Programming Interface) la cual es Keras, y Deep Learning Toolbox Importer for Tensorflow-Keras Models. Para la simulación de los modelos entrenados se usa Simulink de MATLAB.

PALABRAS CLAVE:

- **TENSORFLOW**
- **CONTROL NEURONAL INVERSO**
- **CONTROL CON MODELO DE REFERENCIA LINEAL**

ABSTRACT

In this project, neural network algorithms were developed using the TensorFlow library, for the solution of basic applications such as the XOR problem, which is not linearly separable, the identification of a simple function ($z = x^2 + y^2$) . Through the neural networks that can be trained using this tool, identification, reverse neuronal control and neuronal control are also carried out with a linear reference model of dynamic systems (Antenna with inverted pendulum, Attached Tanks, Beam and Ball). Neural networks were designed and trained in the Python language and then exported to MATLAB R2017b. The networks have been trained using supervised learning and implementing several activation functions in each of their layers, mainly "ReLU". For the development and for the transfer between Python and MATLAB of the trained model, an API (Application Programming Interface) is used which is Keras, and Deep Learning Toolbox Importer for Tensorflow-Keras Models. For simulation of trained models, Simulink from MATLAB is used.

KEYWORDS:

- **TENSORFLOW**
- **REVERSE NEURONAL CONTROL**
- **CONTROL WITH LINEAR REFERENCE MODEL**

CAPÍTULO I

1. INTRODUCCION

1.1 Antecedentes

El control es un campo de rápido desarrollo, complejo y desafiante con gran importancia práctica y potencial. Cuando se usa diversos enfoques de computación de inteligencia artificial, como redes neuronales, probabilidad bayesiana, lógica difusa, aprendizaje automático, computación evolutiva o algoritmos genéticos se introduce el término control inteligente.

El control inteligente describe la disciplina donde se desarrollan métodos de control que intentan emular características importantes de la inteligencia humana en un sistema de cómputo lo que se conoce como inteligencia artificial.

Los avances en inteligencia artificial (IA), informática flexible y campos científicos relacionados han traído nuevas oportunidades y desafíos para que los investigadores enfrenten problemas de sistemas complejos e inciertos, que no pueden resolverse con los métodos tradicionales. Muchos enfoques tradicionales se han desarrollado para la resolución de problemas matemáticos bien definidos con modelos precisos. Sin embargo, normalmente estos métodos carecen de autonomía y no implementan la capacidad de toma de decisiones y, por lo tanto, no pueden proporcionar soluciones adecuadas en entornos inciertos y difusos. Los sistemas inteligentes vienen a representar un nuevo enfoque para abordar esos problemas complejos con incertidumbres y se definen con atributos tales como alto grado de autonomía, razonamiento bajo incertidumbre, mayor rendimiento en la búsqueda de objetivos, alto nivel de abstracción, fusión de datos de una multitud de sensores, aprendizaje y adaptación en un entorno heterogéneo, etc.

Una red neuronal artificial es un sistema de computación que aprende de los datos administrados. Se basa en lo que se conoce del funcionamiento del cerebro humano. Primero, se

genera una estructura de “neuronas” artificiales que se conectan entre sí, y que permiten enviar información entre sí. Luego, se pide que este conjunto resuelva un problema mediante intentos sucesivos, fortaleciendo cada vez las conexiones que conducen al éxito y disminuyendo el fracaso.

Tensorflow qué es un sistema de aprendizaje automático de código abierto para el diseño y aprendizaje de redes neuronales. Ha sido desarrollado por Google Brain Team dentro de la organización de investigación de Machine Intelligence de Google para el aprendizaje automático y la investigación en redes neuronales. Esta herramienta fue liberada por Google convirtiéndose en Open Source a partir del 2015 lo que ha permitido su rápido desarrollo. Es un sistema multiplataforma que opera a gran escala y en entornos heterogéneos.

Tensorflow puede entrenar y ejecutar redes neuronales profundas para la clasificación de dígitos manuscritos, reconocimiento de imágenes, incrustaciones recurrentes, modelos de secuencia a secuencia para traducción automática, procesamiento de lenguaje natural y simulaciones basadas en PDE (Ecuación Diferencial Parcial).

Al ser una herramienta prácticamente nueva y que se está expandiendo de manera acelerada gracias a su flexibilidad para solucionar problemas, se ha utilizado en varios proyectos. Un ejemplo es el “Estudio e implementación de un sistema de control de calidad para la detección de contaminantes superficiales de diferentes tipos de frutas usando visión artificial.”, realizado por Luis Molina y Carlos Corrales, el cual consiste en un problema de clasificación para poder observar de manera automática por medio de visión artificial los contaminantes superficiales de tomates y duraznos.

Las redes neuronales se utilizan en control de calidad tanto de productos como de alimentos, así como también para el área de la salud. El proyecto “Detección de tumores cutáneos malignos y

benignos usando una red neuronal convolucional.", realizado por Jonathan Morocho en EPN sostiene que ayuda a dar un diagnóstico más exacto y prever la evolución de algún tipo de cáncer.

Un proyecto más cercano a lo que se pretende realizar en este trabajo es el: "Diseño e implementación de un sistema de visión artificial para la manipulación, ensamblaje y control de calidad de piezas a través del uso de un brazo robótico para el Laboratorio de Mecatrónica.", realizado por Ney Aucapiña y Brayan Iza, desarrollado en la Universidad de las Fuerzas Armadas – ESPE. En este proyecto se realiza el control de calidad en piezas que se han impreso en 3D con un sistema realimentado que permite que tanto la selección de componentes, así como el diseño e implementación y la impresión sean eficientes y con una alta confiabilidad.

1.2 Justificación

Actualmente Machine Learning es una de las tecnologías más importantes en el mundo. Ha ayudado a resolver problemas que con métodos tradicionales han sido casi imposibles o bien se han tenido resultados poco fiables. Según Gartner, las tecnologías de aprendizaje automático estarán presentes en casi todos los nuevos productos de software en 2020. Es importante que se tenga una idea acerca de esta nueva tecnología que está en constante desarrollo.

La automatización industrial también ha sido impactada por el aprendizaje automático, muchas veces para calibrar un robot industrial para que realice una actividad definida se necesita de muchas calibraciones, pruebas y errores, en cambio si se hace uso de aprendizaje automático, el sistema será más flexible y robusto. Podría cambiar las condiciones de luz, temperatura, etc. En cuanto el robot sabrá cómo actuar de mejor manera.

La carrera de Ingeniería Electrónica en Automatización y Control, tiene una materia llamada "Control Inteligente", en la cual se enseña los fundamentos de lo que es Aprendizaje Automático

de manera tradicional, si bien es importante entender los fundamentos de esto, se necesita llegar un poco más lejos, a lo práctico, utilizando las herramientas de la literatura científica como TensorFlow para aprendizaje profundo. El uso de esta herramienta a través del desarrollo de aplicaciones sencillas y didácticas permitirá generar interés en esta área y contribuir a una formación académica más competente de los estudiantes.

1.3 Alcance del proyecto

Este proyecto se centra en realizar ejemplos de aplicación de la herramienta TensorFlow, que ayuden a comprender por medio de guías de laboratorio usando un computador con Python instalado y la librería de TensorFlow el funcionamiento del aprendizaje automático: desde problemas básicos como entrenamiento de una red neuronal para que realice operaciones lógicas (AND, OR, XOR), clasificación de imágenes, e identificación de números manuscritos hasta resolver el problema de control realimentado de procesos usando redes neuronales de aprendizaje profundo.

Como aplicación final se implementará la simulación del control neuronal realimentado con modelo de referencia lineal y sistema de control con red neuronal inversa para dos problemas clásicos de la teoría de control: sistema de viga y bola, sistema de tanques acoplados.

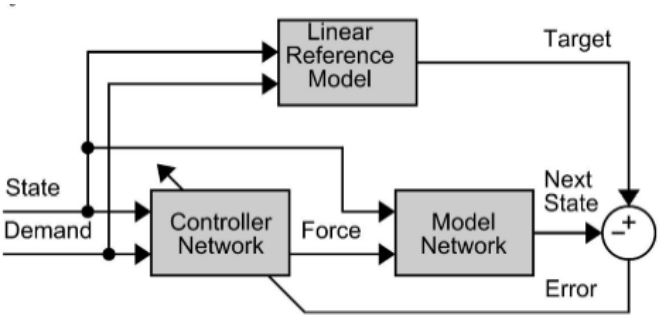


Figura 1. Esquema de MRL

1.4 Objetivos

1.4.1 Objetivo General

Desarrollar aplicaciones didácticas haciendo uso de la librería TensorFlow para resolver problemas de control de procesos realimentados mediante redes neuronales de aprendizaje profundo.

1.4.2 Objetivos Específicos

- Familiarizarse con la librería TensorFlow, sus diferentes funciones y su sintaxis.
- Desarrollar aplicaciones básicas de identificación mediante redes neuronales con aprendizaje profundo.
- Usar un sistema controlado por PID como Modelo de Referencia Lineal.

CAPÍTULO II

2. MARCO TEÓRICO

2.1. Redes Neuronales

Se puede definir a una red neuronal como un conjunto interconectado de nodos de procesamiento simple, cuya funcionalidad se basa en la neurona biológica. La capacidad de procesamiento de la red se almacena en los llamados "pesos" de conexión entre unidades, obtenidos mediante un proceso de aprendizaje con un conjunto de patrones de entrenamiento.(Gurney, 1997)

2.1.1. Neurona biológica

Las neuronas biológicas se comunican a través de señales eléctricas a manera de impulsos o "picos" en el voltaje de la pared celular por medio de la unión conocida como sinapsis. La neurona biológica se compone de unas ramificaciones llamadas dendritas, las cuales son encargadas de recibir información, y de una fibra ramificada conocida como axón que se encarga de enviar información a otras neuronas.(Gurney, 1997)

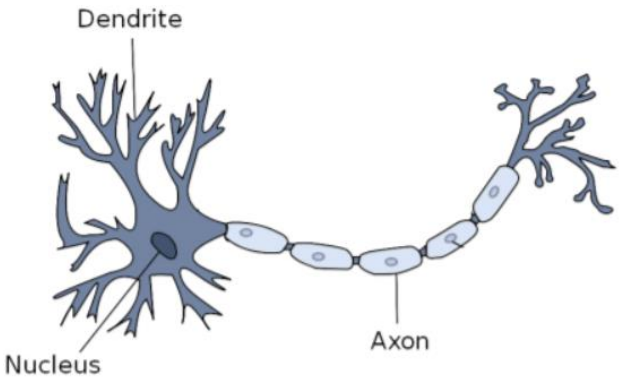


Figura 2. Neurona biológica y sus componentes

Fuente: (Kriesel, 2009)

2.1.2. Neurona artificial

Una neurona artificial intenta replicar la estructura y el comportamiento de una neurona biológica, es decir, recibe señales diferentes x_i, y las procesa de manera predefinida. Dependiendo del resultado del procesamiento, la neurona decide si se activa o no la señal de salida.

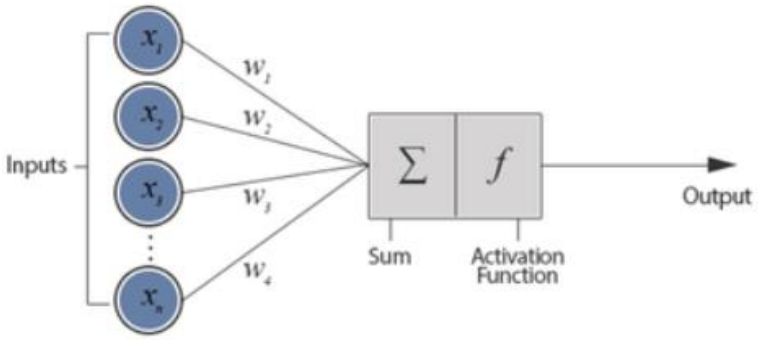


Figura 3. Modelo de una neurona artificial

Fuente: (Kukreja, N, Siddesh, & Kuldeep, 2016)

Los elementos clave de una neurona artificial son los siguientes:

- **Las entradas** de una neurona artificial vendrían a corresponder a las dendritas de una neurona biológica, las cuales se encargarían de recibir datos de otras neuronas.
- **Los pesos sinápticos W_{ij}** son los que se modificaran durante el entrenamiento de la red neuronal junto con el factor de importancia (bias). Estos valores permitirán que la red sirva para un propósito u otro.
- **Una regla de propagación.** Para obtener el valor potencial postsinaptico (valor que está en función de las entradas y de los pesos) se realiza algún tipo de operación entre las entradas y los pesos sinápticos. Una de las operaciones más comunes es sumar las entradas teniendo en cuenta la importancia de cada una (peso). A esto se lo llama suma ponderada y está definida por:

$$h_i(t) = \sum_{i=0}^{n} x_i * W_i$$

- **Una función de activación.** El valor que se adquiere con la regla de propagación, se filtra a través de la función de activación, siendo esta la que nos da la salida de la neurona. Dependiendo del problema para el que se desee entrenar una red neuronal se escoge diferentes funciones de activación.

En la tabla 2 se listan más funciones de activación. Una de las cuales es la ReLU que es la función que por defecto aparece en las redes neuronales de aprendizaje profundo.

2.1.3. Redes neuronales de aprendizaje profundo

El aprendizaje profundo tiene muchas capas ocultas y también permite utilizar muchos más parámetros antes de que se produzca un ajuste excesivo.

El aprendizaje profundo es un algoritmo extendido de redes neuronales convencionales, donde el número de capas ocultas y el número de neuronas son más que los de las redes neuronales convencionales.

2.1.4. Estructura de una red neuronal

La estructura de una red neuronal está compuesta por:

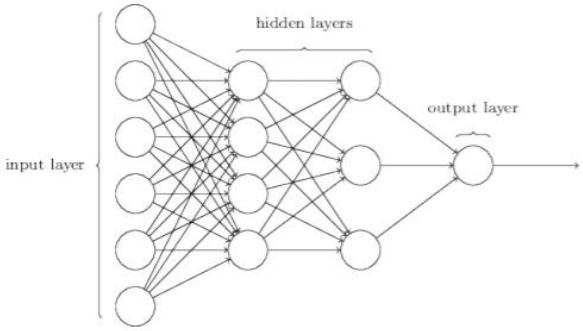


Figura 4. Estructura de una red neuronal

Capa de entradas. Capa que recibe las señales de entrada.

Capa(s) oculta(s). Un conjunto de neuronas que se ubica entre la capa de entrada y capa de salida permitiendo a la red más grados de libertad y otorgando mayor capacidad de procesamiento. Este conjunto puede ser conformado por una o múltiples capas.

Capa de salida. En general esta suele ser conformada por una sola neurona, siendo su salida de forma binaria (0 o 1). Pero también puede conformarse con más que una sola neurona y tener salidas con datos enteros o flotantes según el problema a resolver.

2.1.5. Tipos de entrenamiento

Los algoritmos de aprendizaje lo que buscan es reducir una función de coste o pérdida, a través de métodos numéricos iterativos. El método más popular para entrenar una red neuronal es la retropropagación.

La retropropagación, abreviatura de "propagación de errores hacia atrás", es un algoritmo para el aprendizaje supervisado de redes neuronales artificiales utilizando el descenso de gradiente. Dada una red neuronal artificial y una función de error, el método calcula el gradiente de la función de error con respecto a los pesos de la red neuronal.

Existen dos tipos de entrenamiento: Supervisado y no supervisado

Entrenamiento supervisado

Este método es el más utilizado y popular. Los datos para el entrenamiento incluyen las entradas como las salidas deseadas (etiquetas) de modo que una función pueda calcular el error para una predicción dada. La supervisión viene a ser cuando se realiza una predicción durante el entrenamiento y se evalúa el error y se altera los pesos de la red neuronal. (Tim Jones, 2017)

Este método de aprendizaje implica la creación de una función que puede ser entrenada mediante el uso de un conjunto de datos de entrenamiento conocidos, y luego se aplica un conjunto de datos no conocido para la red y así poder cumplir un rendimiento predictivo. (Tim Jones, 2017)

La variación de los pesos en una etapa $(m+1)$-ésima, se daría de la siguiente manera:

$$\omega_{ij}^{m+1} = \omega_{ij}^{m} + \Delta\omega_{ij}^{m}$$

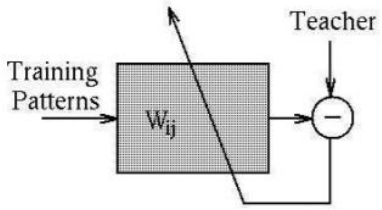


Figura 5. Aprendizaje supervisado

Entrenamiento no supervisado

En este método el conjunto de datos de aprendizaje no incluye el resultado deseado, por lo tanto, no habría forma de supervisar la función y de ahí el nombre. En cambio, la función lo que intenta es clasificar el conjunto de datos en "clases" para que cada clase contenga una parte del conjunto de datos con características comunes, es decir, a través de experiencias recogidas de los patrones de entrenamiento anteriores. (Marín Diazaraque, 2007)

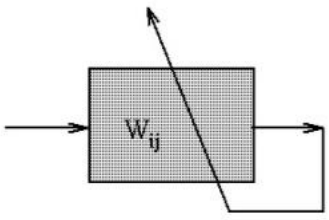


Figura 6. Aprendizaje no supervisado

2.2. Descripción de la librería Tensorflow

Una de las librerías de aprendizaje profundo más desarrollada en la actualidad desde que fue liberada ha sido Tensorflow de Google. Esta librería se usa constantemente en el aprendizaje automático para mejorar las principales aplicaciones de Google como son: el motor de búsqueda, la traducción simultánea, generación de subtítulos automáticos en videos de YouTube, etc.

Esta biblioteca ha sido desarrollada para construir y entrenar redes neuronales. En función de la estructura de capas y neuronas que la conforman se puede representar desde un simple modelo de regresión hasta estructuras más complejas de machine learning. Esta herramienta se puede ejecutar en múltiples CPU o GPU, sistemas operativos móviles, y se lo puede compilar en varios lenguajes de programación como Python, C++ o Java.

Existen varias API (Application Programming Interface), que se ejecutan sobre TensorFlow: TFlearn, Keras, que permiten que la programación de los algoritmos sea más amigable con el usuario, sea modular, configurable y fácil de extender.

Una API (Application Programming Interface) es un intermediario de software que permite que dos aplicaciones se comuniquen entre sí.

Keras es una API de alto nivel de TensorFlow para construir y entrenar modelos de aprendizaje profundo que está escrita en Python y que es capaz de ejecutarse sobre TensorFlow.

En este proyecto se usará Keras, ya que esta nos permitirá realizar la exportación del modelo en un archivo (.h5) que es compatible con MATLAB, el cual se usará para realizar las simulaciones.

2.2.1. Características de Tensorflow

- Realiza efectivamente expresiones matemáticas entre matrices muldimensionales.
- Tiene soporte para redes neuronales profundas y conceptos de aprendizaje profundo.
- Ejecuta los algoritmos en GPU/ CPU
- Permite alta escalabilidad de computo entre máquinas y grandes conjuntos de datos.
- Se ejecuta en diferentes plataformas:
    - Windows, macOs o Linux
    - Servicios Web como Cloud
    - Dispositivos móviles como iOS y Android.

2.2.2. Red neuronal en TensorFlow

- El entrenamiento de la red neuronal consta de tres partes:

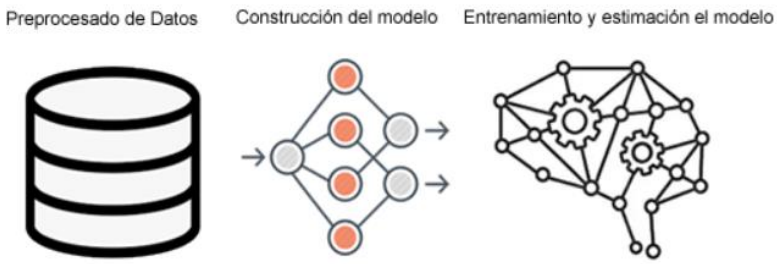


Figura 7. Secuencia de entrenamiento de red neuronal

- **Pre procesado de datos**

  Para el entrenamiento de una red neuronal se necesita de un conjunto de patrones de entrenamiento. Datos que pueden ser recolectados de manera manual o a su vez de manera digital usando aplicaciones específicas configuradas previamente para este propósito.

  La importación del conjunto de patrones de entrenamiento en este proyecto se lo hará por medio de Numpy en Python como se muestra a continuación:

```
import numpy as np
Pm = np.matrix(pd.read_csv("InMRLAntena.csv", header=None).values)
Tm = np.matrix(pd.read_csv("OutMRLAntena.csv", header=None).values)
```

Figura 8. Importación del conjunto de patrones de entrenamiento

- **Construcción del modelo**

  Para esto se define el número de neuronas de entrada, el número de capas ocultas, el número de neuronas en cada capa oculta y el número de neuronas en la capa de salida según sea el problema a solucionar.

- **Entrenamiento y estimación el modelo**

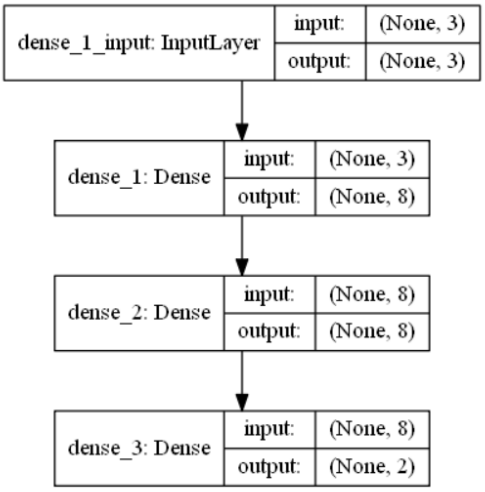
  Se define el número de épocas, el optimizador y la función de pérdida.

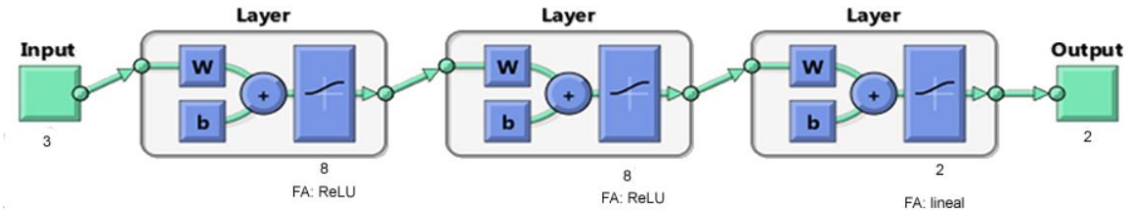
2.2.3. Tipos de modelos Redes Neuronales

Existen dos tipos de modelos que se pueden construir: modelo secuencial y modelo funcional.

Modelo Secuencial

Este modelo se compone de un conjunto de capas lineales. En este modelo se necesita saber la dimensión de entrada. Por esta razón, la primera capa en un modelo secuencial (y solo la primera, porque las siguientes capas pueden hacer inferencia de forma automática) necesita recibir información sobre su dimensión de entrada.



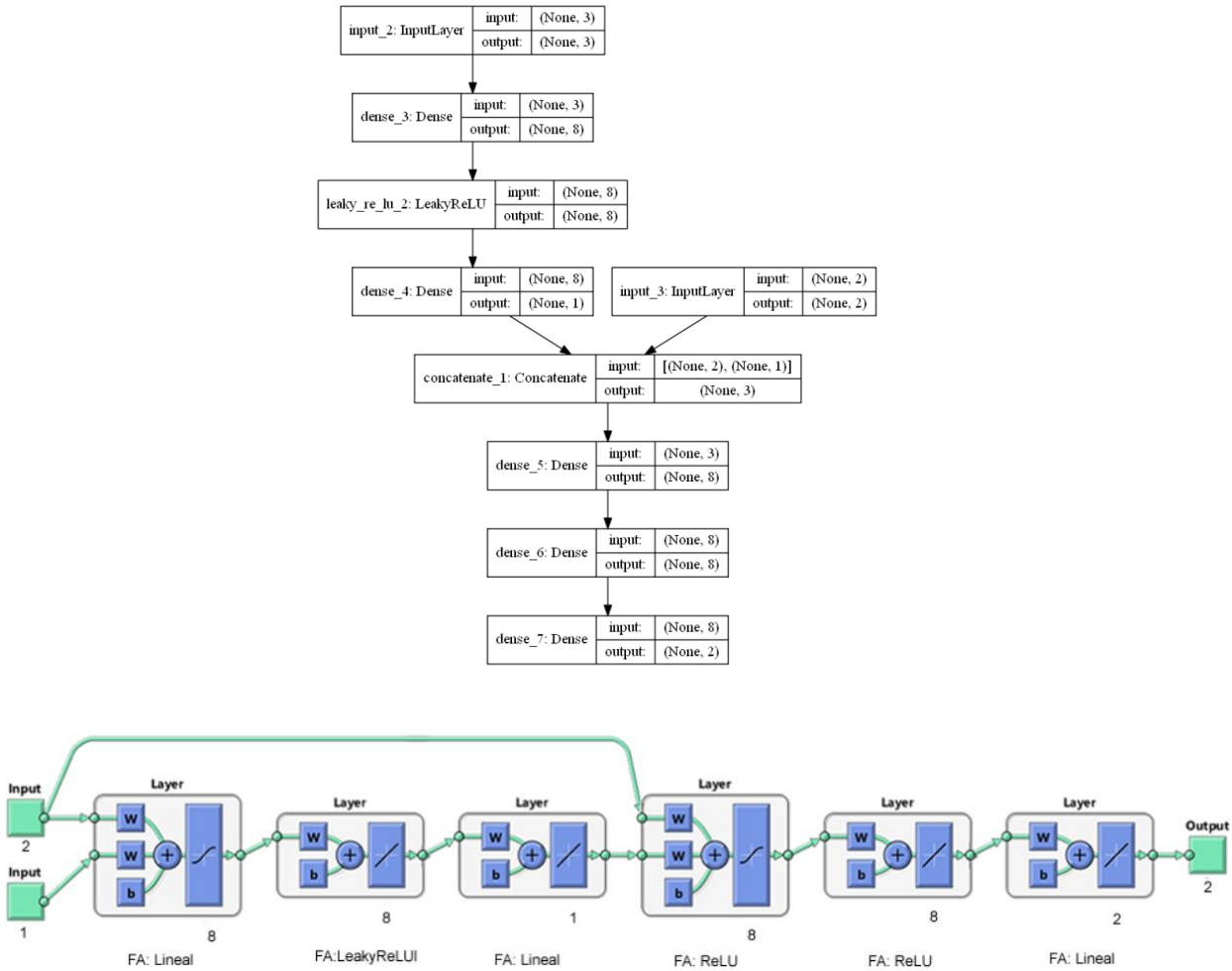



```
#Importacion de librerias necesarias
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
#Modelo Secuencial
model = Sequential()
#Se añade una capa de 32 neuronas con activacion 'relu', la cual
#se conecta a la capa de entrada de dimension 3
model.add(Dense(8, input_dim=3,activation='relu'))
model.add(Dense(8,activation='relu'))
#Se añade varias capas de manera secuencial
model.add(Dense(2))
```

Figura 9. Ejemplo de estructura de Modelo Secuencial

Modelo Funcional

Este permite definir modelos más complejos, ya sea con múltiples salidas, gráficos acíclicos dirigidos o modelos con capas compartidas. En este modelo se puede definir la conexión de las capas, que capas son de entrada y de salida.




```
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.layers import Flatten
from keras.models import Model
from keras.utils import plot_model
#Controlador MRL
visible1 = Input(shape=(3,))
# Se especifica que capa se une a cada capa
# Capa visible1 se conectara con dense11
dense11 = Dense(8,input_shape=X_train.shape)(visible1)
Leaky11 = LeakyReLU(alpha=0.085)(dense11)
dense12 = Dense(1)(Leaky11)
visible2 = Input(shape=(2,))
# Unir entrada de red neuronal y estados
merge = concatenate([visible2, dense12])
# ModelNetwork
hidden1 = Dense(8, activation='relu',weights=[Pc1,Bc1],trainable=0,input_shape=merge.shape)(m
erge)
hidden2 =(Dense(8,activation='relu',weights=[Pc2,Bc2],trainable=0))(hidden1)
output = Dense(2,weights=[Pc3,Bc3],trainable=0)(hidden2)
model = Model(inputs=[visible1, visible2], outputs=output)
```

Figura 10. Ejemplo de estructura de Modelo Funcional

2.2.4. Funciones principales

Dense

Tanto en el modelo secuencial como en el modelo funcional es necesario especificar qué tipo de capas se desea añadir, cuantas neuronas y que función de activación, para esto se usa:

En Modelo Secuencial.

```
model.add(Dense(units, input_dim=num,weights=[P,B],trainable=0, activation='relu'))
```

Figura 11. Sintaxis de función Dense para modelo secuencial

Argumentos:

units. Se especifica el número de neuronas para esa capa.

input_dim. Se especifica la dimensión del tensor de entrada.

weights. Si se desea especificar los pesos de las neuronas de una capa.

trainable. Se especifica si se desea que los pesos de esa capa aprendan o no.

activation. Se introduce el nombre de la función de activación que se desea para esa capa.

trainable. Se especifica si se desea que los pesos de esa capa aprendan o no.

En Modelo Funcional

```
nomCS = Dense(units, activation='relu',weights=[P,B],trainable=0,input_shape=num)(nomCAN)
```

Figura 12. Sintaxis de función Dense para modelo funcional

Argumentos:

units. Se especifica el número de neuronas para esa capa.

input_shape. Se especifica la dimensión del tensor de entrada.

weights. Si se desea especificar los pesos de las neuronas de una capa.

trainable. Se especifica si se desea que los pesos de esa capa aprendan o no.

activation. Se introduce el nombre de la función de activación que se desea para esa capa.

NomCS Nombre de la capa actual.

nomCAN. Nombre de la capa que se desea conectar con NomCS

A continuación, se muestra una tabla donde se enlista las funciones básicas y avanzadas de activación. De las cuales las que en este proyecto se han usado son: tanh, relu, LeakyReLU.

Tabla 1

Lista de funciones de activación

Funciones de Activación		Funciones Avanzadas
• elu • sofmax • selu • softplus • softsign • relu • tanh	• sigmoid • hard_sigmoid • exponential • linear	• LeakyReLU • PReLU • ELU • ThresholdedReLU • Softmax • ReLU

Fuente: Keras Documentation

Función Rectified Lineal Unit (relu), transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran.(Calvo, n.d.)

La ecuación de la función “relu” es:

$$f(x) = \max(0, x) = \begin{cases} 0 \; para \; x < 0 \\ x \; para \; x \geq 0 \end{cases}$$

Función Leaky ReLU (LeakyReLU), es una versión mejorada de la función relu, que evita que en el entrenamiento al actualizar los pesos provoque que dicha neurona nunca se llegue a activar en ningún punto de los datos nuevamente, ya que su gradiente siempre será igual a cero.

Para esto, en lugar de que la función sea cero cuando x <0, una LeakyReLU tendrá una pequeña pendiente negativa (de 0.01, más o menos).

La ecuación de la función “LeakyReLU” es:

$$f(x) = \begin{cases} \alpha x \; para \; x < 0 \\ x \; para \; x \geq 0 \end{cases}$$

Donde,

α*, es una pequeña constante*

Función tangente hiperbólica (tanh), ya que esta transforma los valores introducidos en una escala (-1,1), donde los valores altos tienen la manera asintótica a 1 y los valores bajos tienen la manera asintótica a -1. Esta función en especial se usa en problemas donde el resultado es binario, es decir, tiene solo dos opciones de resultado. (Calvo, n.d.)

La ecuación de la función “tanh” es:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Función sigmoide (sigmoid), además de ser la función más conocida de activación de neuronas en redes neuronales, esta se caracteriza por su salida que está acotada entre 0 y 1, tiene lenta convergencia y un excelente rendimiento en la última capa.(Calvo, n.d.)

La ecuación de la función “sigmoid” es:

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

Compilación.

Antes de realizar el entrenamiento del modelo, se necesita definir el proceso de aprendizaje, esto se realiza a través del método compile.

```
model.compile(optimizer ,loss='none',metrics=['none'])
```

Figura 13. Sintaxis de función Compile

Argumentos:

Optimizer. Este parámetro se encarga de evaluar cuanto y de qué manera corregir los pesos a lo largo del aprendizaje, es decir, define el ritmo de aprendizaje. Debe ser un String (nombre del optimizador)

Loss. También llamada función de coste, trata de determinar el error entre el valor real y estimado, con el fin de optimizar los parámetros de la red neuronal. Debe ser un String (Nombre de la función de coste).

Metrics: Son funciones que ayudan al monitoreo del proceso de entrenamiento, este nos ayuda a observar de mejor manera la evolución de nuestra red neuronal. Debe ser un String (Nombre de la métrica para evaluar el modelo mientras se entrena la red neuronal)

A continuación, se muestra una tabla donde se enlista los diferentes optimizadores, funciones de coste y métricas que se han implementado en la librería TensorFlow. De las cuales las que en este proyecto se han usado son: Optimizador (Adam), Funcion de Coste (mean_square_error) y Metrica (accuracy).

Tabla 2

Lista de optimizadores, funciones de coste y métricas para la función 'compile'

Optimizer	Losses	Metric
• SGD • RMSprop, • Adagrad, • Adadelta, • Adam, • Adamax, • Nadam	• mean_squared_error • mean_absolute_error • mean_absolute_percentage_error • mean_squared_logarithmic_error • squared_hinge • hinge • categorical_crossentropy • sparse_categorical_crossentropy • binary_crossentropy • is_categorical_crossentropy	• accuracy • binary_accuracy • categorical_accuracy • sparse_categorical_accuracy • top_k_categorical_accuracy • sparse_top_k_categorical_accuracy • cosine_proximity • clone_metric

Fuente: Keras Documentation

El **error cuadrático medio (mean_squared_error)**, mide el promedio de los errores al cuadrado, es decir:

$$\mathrm{ECM} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2.$$

Donde,

n, número de datos

$\hat{Y}_i$, salida estimada

Y_i, salida verdadera

La **entropía cruzada binaria (binary_crossentropy)**, es una función de costo de precisión para variables binarias.

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(p_i + (1-y)\log(1-pi))$$

El **descenso estocástico de gradiente (sgd)**, viene a ser una aproximación estocástica del descenso de gradiente, que se usa para minimizar la función objetivo. Este optimizador trata de encontrar mínimos o máximos por iteración. Se hace uso de este optimizador cuando existe muchos ejemplos de entrenamiento, ya que si se hace uso de todos los ejemplos para producir una actualización de los pesos esto es conlleva un costo computacional muy alto.

$$\theta = \theta - \eta * \nabla_{\theta} J(\theta; x, y)$$

Donde,

θ, pesos o bias.

η, factor de aprendizaje.

∇, gradiente.

J, funcion objetivo.

Estimación de momento adaptativo (Adam), Es un método eficiente de optimización estocástica de primer orden, que combina las ventajas de los otros optimizadores de descenso de gradiente estocástico, los cuales son:

Adaptive Gradient Algorithm (AdaGrad), que mantiene una tasa de aprendizaje por parámetro que mejora el rendimiento en problemas con gradientes dispersas (por ejemplo, problemas de lenguaje natural y visión por computadora).

Root Mean Square Propagation (RMSProp), que también mantiene tasas de aprendizaje por parámetro que se adaptan según el promedio de las magnitudes recientes de los gradientes para el peso (por ejemplo, qué tan rápido está cambiando). Esto significa que el algoritmo funciona bien en problemas transitorios y no estacionarios (por ejemplo, ruido).

Las ventajas de usar Adam en problemas de optimización son:

- Fácil de implementar.
- Eficiencia computacional.
- Requerimientos de memoria mínimos.
- Adecuado para problemas con datos grandes de entrenamiento.
- Apropiado para objetivos no estacionarios.
- Apropiado para problemas con gradientes muy ruidosos.

Adam es recomendado actualmente como método de optimización predeterminado para aplicaciones de aprendizaje profundo ya que, al ser de primer orden, su derivada tiene un menor costo computacional en comparación a otros métodos (Levenberg Marquardt y Hessian), que sus derivadas son de segundo orden y por lo tanto requieren mayor costo computacional.

Entrenamiento

Para realizar el entrenamiento de la red neuronal se usa la función *fit*, la cual tiene la siguiente sintaxis:

```
model.fit(x=None, y=None, epochs=1, batch_size=32)
```

Figura 14. *Sintaxis de función 'fit'*

Argumentos:

x: Patrones de entrenamiento de entrada (array, matriz).

y: Patrones de entrenamiento de salida (array, matriz).

epochs: Es un entero y es el número de épocas a entrenar la red neuronal.

batch_size: Es un entero. Numero de ejemplos por gradiente a actualizarse.

Predicción

Para realizar una predicción con una entrada de datos especifica usa la función *predict,* se debe tomar en cuenta la dimensión de la matriz o array de entrada.

```
model.predict(x)
```

Figura 15. *Sintaxis de función 'predictt'*

Argumentos:

x: Datos de prueba con la que se desea predecir (array, matriz).

Imprimir resumen del modelo

Para imprimir el resumen de modelo en el cual se puede apreciar de cuantas capas y que función de activación tiene cada una, se usa la función *summary.*

```
print(model.summary())
```

Figura 16. *Sintaxis de función 'summary'*

Argumentos:

model: Nombre del modelo a imprimir.

2.3. Herramienta gráfica

Para exportar el modelo desde Python a una imagen de formato png en una carpeta local se usa el siguiente código. Se necesita instalar las librerías pydotplus y pydot perteneciente a la API Keras.

```
1.  import tensorflow
2.  from tensorflow import keras
3.  import pydotplus
4.  from keras.utils.vis_utils import model_to_dot
5.  from keras.utils.vis_utils import plot_model
6.  keras.utils.vis_utils.pydot = pydot
7.  plot_model(model, to_file='model_plot.png', show_layer_names=True)
```

Figura 17. *Código para exportar el modelo a una imagen png*

Argumentos:

model: Nombre del modelo a exportar.

to_file: Nombre con el que se desea exportar como imagen el modelo.

Show_layer_names: Si se desea exportar junto a la imagen el nombre de las capas de la red neuronal.

2.4. Componentes de TensorFlow

En la siguiente figura, se puede apreciar los principales componentes de TensorFlow, los cuales se describen a continuación.

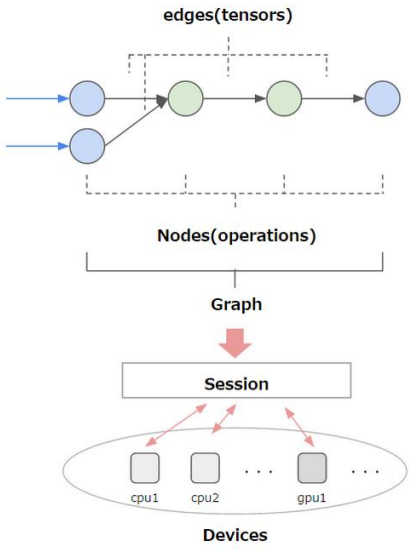


Figura 18. Componentes de TensorFlow

Fuente: TensorFlow

2.4.1. Tensor

Un tensor es una entidad matemática que tiene las siguientes propiedades:

- Tipo de dato, por ejemplo (float32, int32 o string)
- Es un vector o una matriz de n dimensione

Cada elemento en un Tensor es del mismo tipo de dato y este tipo de dato debe ser conocido.

2.4.2. Nodos

Cada nodo en un gráfico representa una instancia de una operación matemática (suma, división o multiplicación).

Un **tensor** se puede generar a partir de datos de entrada o ser el resultado de un cálculo. En TensorFlow, todas las operaciones se realizan dentro de un **grafo**.

2.4.3. Grafo

Un **grafo** es un conjunto de cálculos que se realiza de manera sucesiva. Los grafos reúnen y describen todos los cálculos de la serie realizados durante el entrenamiento. Un grafo se puede guardar para poder usarlo en un futuro.

Para poder explicar lo que es un grafo es exponiendo un ejemplo simple, para poder escribir un código para la función $f(x, y) = x^2y + 2 + y$. El Grafo en TensorFlow sería el siguiente:

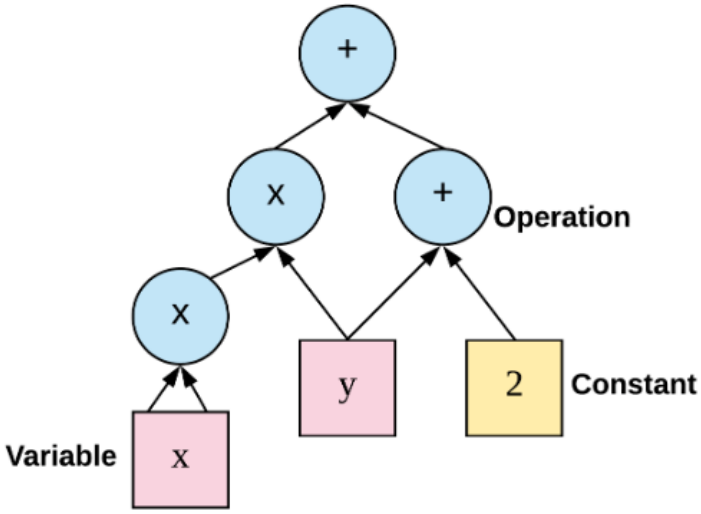


Figura 19. Esquema de un grafo en TensorFlow

Fuente: TensorFlow

2.4.4. Sesión

Una sesión permite ejecutar gráficos o parte de gráficos. Asigna recursos (en una o más máquinas) y mantiene los valores reales de resultados intermedios y variables.

2.4.5. Dispositivos

Dispositivos donde se puede ejecutar en múltiples CPU o GPU, incluso en un sistema operativo móvil.

2.5. Sistemas Dinámicos

Un sistema dinámico tiene relación con la evolución de algo a lo largo del tiempo. Se define como un modelo que describe la evolución temporal de un sistema, cuyas variables describen el estado en cualquier instante, y que se rige a una regla dinámica que especifica el futuro inmediato de las variables de estado, dado los valores presentes de esas variables.(Meiss, 2007)

2.5.1. Identificación de Sistemas Dinámicos

Los modelos de sistemas reales prácticamente están presentes en todas las disciplinas. Estos pueden ser útiles para el análisis del sistema, es decir, para una mejor comprensión del mismo. Los modelos permiten predecir o simular el comportamiento de un sistema.(Nelles & Nelles, 2001)

La mayoría de los sistemas reales presentan un comportamiento dinámico no lineal. Un modelo lineal solo puede describir el sistema para un pequeño rango de entradas y salidas.(Verdult, 2002)

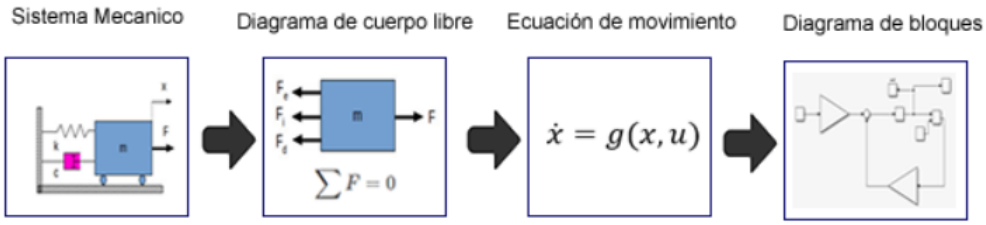


Figura 20. Representación de modelo dinámico

Para realizar la identificación de un modelo dinámico, primero se obtiene la ecuación dinámica que gobierna dicho sistema y que se puede representar de la siguiente manera:

$$\dot{x} = g(x,u)$$

Ya definida la ecuación dinámica, esta se puede representar como un diagrama de bloques, del cual se puede simular inyectando valores de las variables de estado, obteniendo en su salida las variaciones de las mismas.

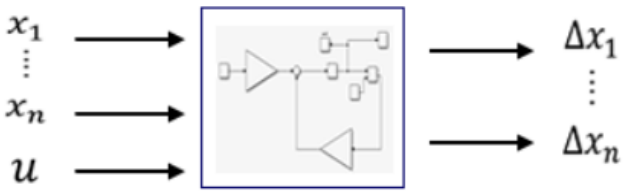


Figura 21. Obtención de variaciones de estado

Posterior, se realiza el entrenamiento de una red neuronal, la cual tendrá como salidas las variaciones de estado y como entradas las condiciones iniciales y la señal de control. El cual se debe comportar como el modelo identificado.

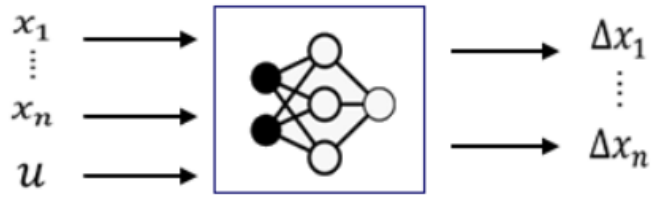


Figura 22. Diagrama identificación

Una red neuronal permite solamente identificar una función estática. Por esta razón el sistema dinámico se identifica mediante los estados iniciales y la variación que se produce luego de un tiempo de discretización “Ts” en esos estados.

Para convertir el sistema estático de la red neuronal a un sistema dinámico se utiliza un retardo de primer orden que permite obtener el valor de la señal luego de un retardo.

La red neuronal recibe como entrada el estado actual y calcula la variación de ese estado. El estado actual sumado con la variación produce el estado siguiente. El sistema se vuelve dinámico cuando se alimenta el estado siguiente como el estado actual a través de un retardo de primer orden.

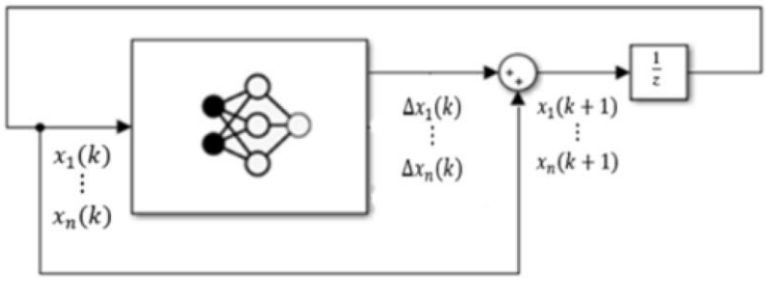


Figura 23. Señal de salida de red neuronal con retardo de primer orden

2.5.2. Control Neuronal Inverso

El control por modelo inverso es una técnica, que busca cancelar la dinámica de la planta al colocar un elemento en cascada con ella, en este caso una red neuronal, siendo este una aproximación matemática del inverso de la planta. De esta manera se busca que la salida sea lo más parecida posible a la referencia.

Obtenido el conjunto de variaciones que se producen en los estados, más las condiciones iniciales y la señal de control. Para el entrenamiento de un controlador neuronal inverso, se considera el conjunto de datos de manera distinta a la identificación. Los estados, las variaciones de estados se tomarán como entradas y la señal de control se tomará como salida.

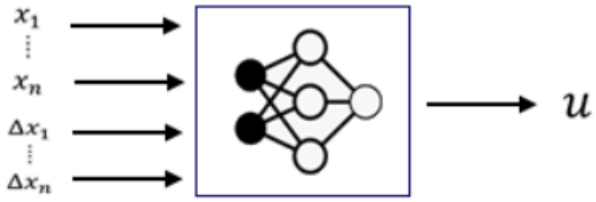


Figura 24. Diagrama controlador neuronal inverso

Uniendo el controlador neuronal inverso y la planta quedaría:

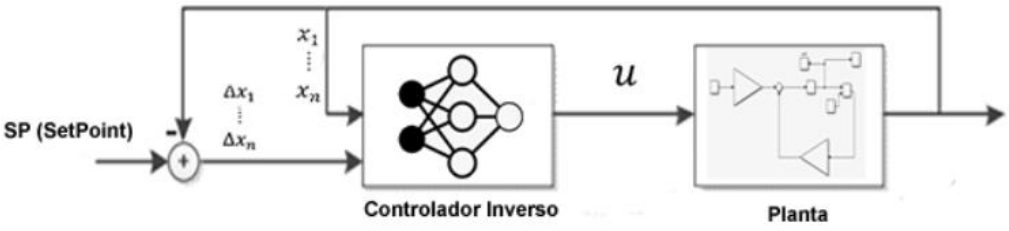


Figura 25. Esquema de control con red neuronal inversa

2.5.3. Control Neuronal con modelo de referencia

La arquitectura de control de referencia del modelo utiliza dos redes neuronales: una red destinada a ser el controlador y una red del modelo de la planta previamente entrenada. Teniendo la red de la planta ya identificada, se procede a entrenar el controlador para que la salida de la planta siga la salida del modelo de referencia.

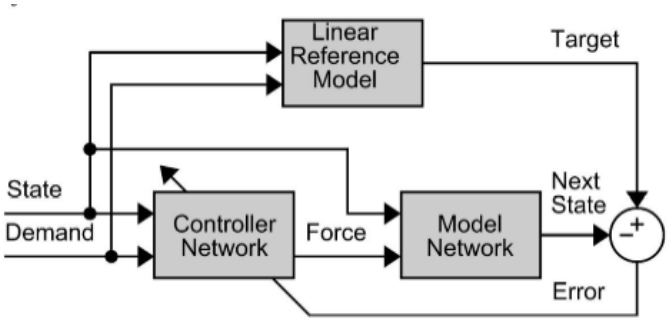


Figura 26. Esquema de control con red neuronal con modelo de referencia

Al momento del entrenamiento de la red se debe tomar en cuenta que no se deben modificar los pesos de la red ya identificada, sino que por medio de la retro propagación se modifiquen solamente los pesos de la red del controlador.

2.6. Instalación de software necesario.

2.6.1. Anaconda Python Distribution

Se usará la Suite de Anaconda que nos facilitará la instalación del ambiente que es necesario e incluirá Jupyter Notebooks, que es una aplicación que ayuda a hacer los ejercicios en Python paso a paso.

Descarga e Instalación

Primero se descarga el paquete de instalación de la página principal de Anaconda, se elige la plataforma y la versión.

Para este proyecto se ha utilizado la plataforma Windows y la versión Python 3.7.

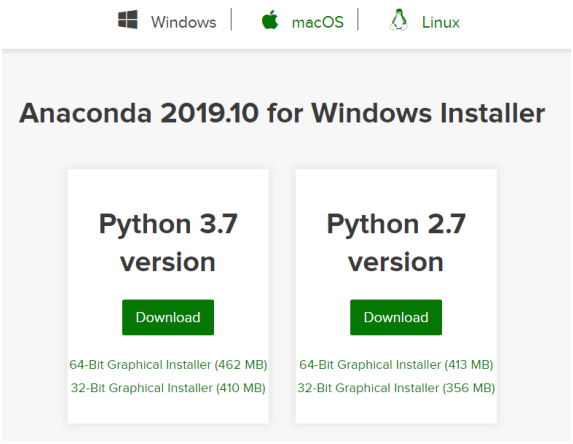


Figura 27. Página de descarga de Anaconda

Luego de haber descargado, se procede a la instalación otorgando permisos de Administrador.

Anaconda Navigator

Anaconda viene con una interfaz gráfica de usuario (GUI) de escritorio llamada Anaconda Navigator, que permite iniciar aplicaciones y administrar fácilmente paquetes, entornos y canales. Abriremos la aplicación y veremos una pantalla similar a esta:

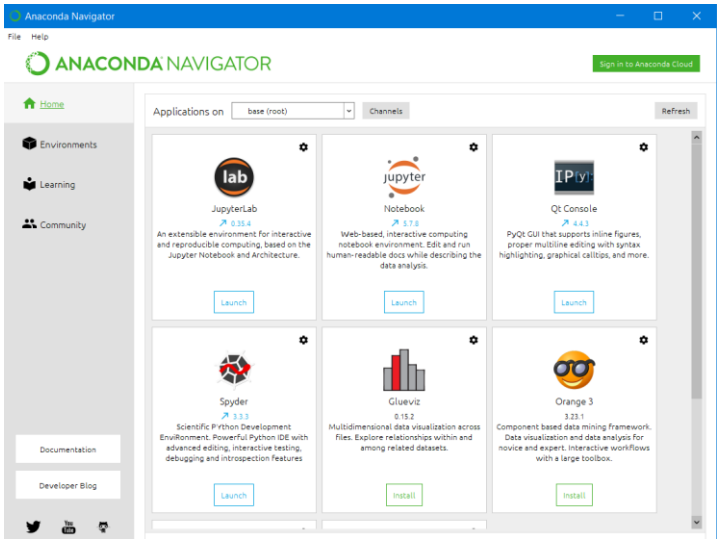


Figura 28. Interfaz gráfica de Anaconda Navigator

Creación de nuevo Ambiente

Al hacer esto nos permitirá tener un ambiente desde cero, solo con las librerías básicas y nos permitirá instalar lo que necesitemos. Esto nos ayuda para que, si hay error de librerías o hay un error desconocido, simplemente podamos eliminar el ambiente y crear uno nuevo, en vez de desinstalar Anaconda y volverlo a instalar.

Iremos al apartado **Environments**, luego a **Create**, introduciremos el nombre con el que deseemos llamar a nuestro nuevo ambiente de trabajo y la versión de Python a usar. En este proyecto se ha usado la versión Python 3.7.

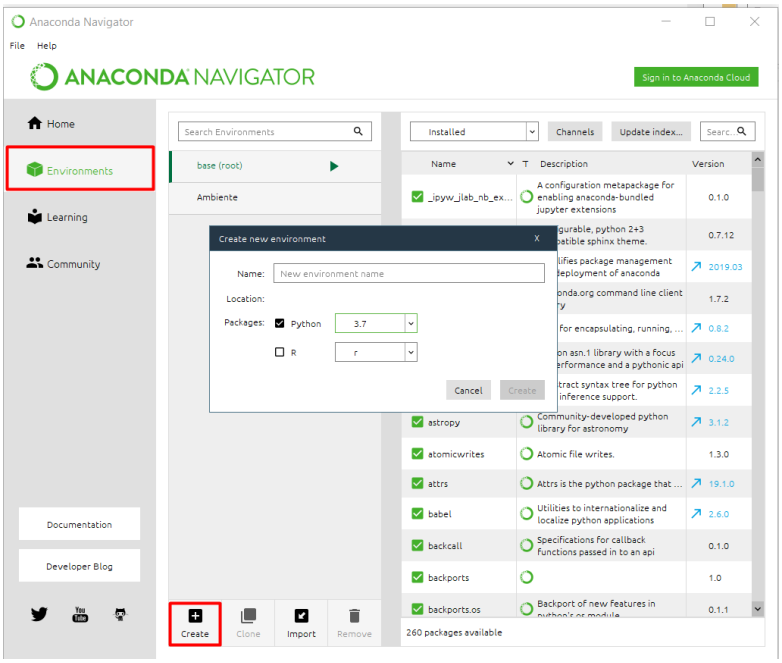


Figura 29. Creación de un nuevo Ambiente

Luego de la creación, volveremos al apartado Home y seleccionaremos el nombre del Ambiente, que en este caso el nombre es “Ambiente” e instalaremos la aplicación Jupyter Notebook, como se muestra:

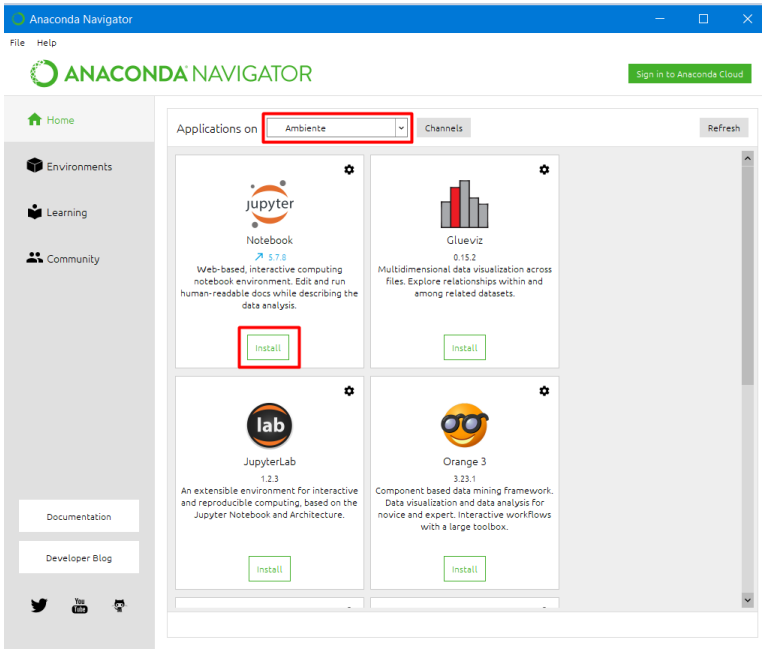


Figura 30. Instalación de Jupyter Notebook

Instalación de librerías

En este caso haremos uso de la consola de Anaconda (Anaconda Prompt), para la instalación de los paquetes, ya que nos permite hacerlo de una manera más específica.

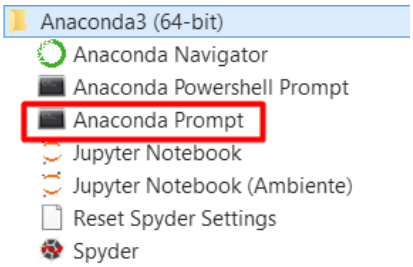


Figura 31. Anaconda Prompt

Al iniciar la consola, deberemos activar el Ambiente que se creó anteriormente, para este caso sería:

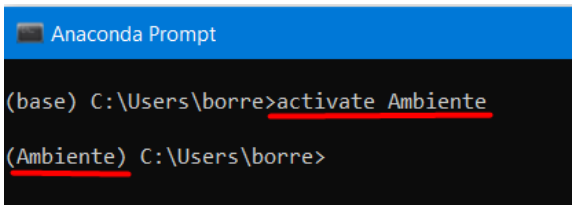


Figura 32. Activación del Ambiente en consola Anaconda Prompt.

Luego de activar el Ambiente, procedemos a instalar los paquetes necesarios que están listados a continuación.

```
1.  pip install tensorflow
2.  pip install keras
3.  pip install numpy
4.  pip install sklearn
5.  pip install pydot
6.  pip install pydotplus
7.  pip install matplotlib
```

Figura 33. Comandos para instalar librerías necesarias.

Ya instalado los paquetes necesarios ya prodemos ejecutar la aplicación Jupyter Notebook, la cual abrirá nuestro navegador predeterminado y tendremos una pantalla como la siguiente:

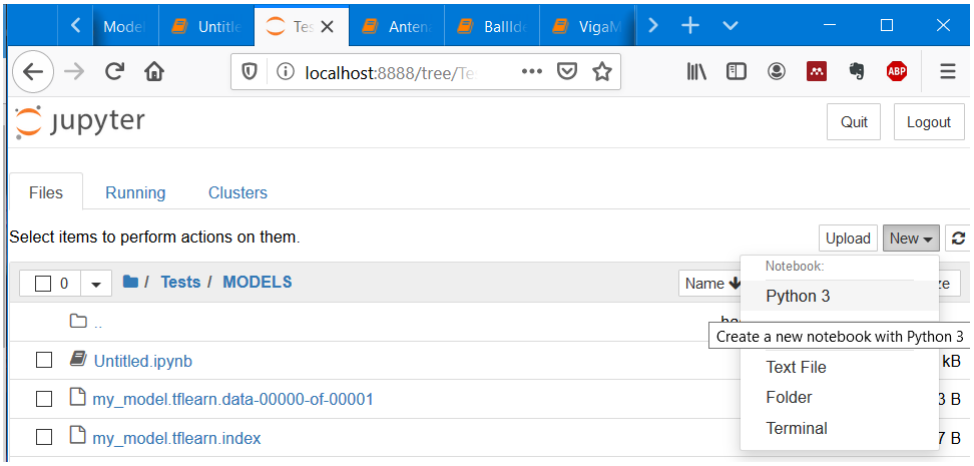


Figura 34. Intefaz de Jupyter Notebook

Ya en Jupyter Notebook, podemos crear un nuevo archivo con Python 3, importar las librerías necesarias y empezar a escribir y ejecutar paso a paso el código.

Importación de librerías

Se realiza la importación de las librerías necesarias por medio del siguiente código:

```
#Identificacion Antena
import numpy as np
import tensorflow
import matplotlib.pylab as plt
import from tensorflow import keras
import pydot
import sklearn
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LeakyReLU
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
```

Figura 35. Importación librerías – Identificación Antena

Toolbox necesario para importación del modelo a MATLAB

Para poder importar el modelo a MATLAB, primero se instala el paquete (Deep Learning Toolbox Importer for Tensorflow-Keras Models).

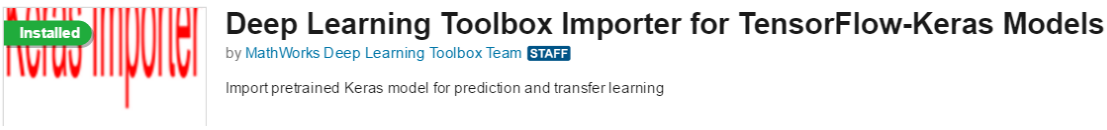


Figura 36. DLTI for TensorFlow-Keras Models

CAPÍTULO III

3. DESARROLLO DE ALGORITMOS BASICOS Y CONTROL REALIMENTADO

La carrera de Ingeniería Electrónica en Automatización y Control, tiene una materia llamada "Control Inteligente", en la cual se busca utilizar herramientas de la literatura científica como es la librería de aprendizaje profundo TensorFlow. En este capítulo se especifica como tener un ambiente adecuado para la realización de aplicaciones básicas y complejas.

3.1. Ejemplos de Aplicaciones Básicas

3.1.1. Problema XOR

Uno de los problemas más conocidos que no es separable linealmente con una red neuronal es la función OR exclusiva o XOR. Es necesario utilizar una red neuronal de al menos dos capas que permita la separabilidad no lineal de la compuerta XOR. Esta función lógica normalmente tiene dos entradas y una salida que depende del valor de las entradas. 0 o 1 (Verdadero o Falso).

Elaboración de la red neuronal

Patrones de entrenamiento

La tabla de verdad para dos entradas x,y para la función XOR es la siguiente:

Tabla 3
Tabla de la Función XOR

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Para poder ingresar estos datos a nuestra red neuronal es necesario tratarlos como arreglos que en este caso “X” será de 4 filas, 2 columnas y “Y” será de 4 filas, 1 columna. Como se muestra a continuación:

```
1.  X = [[0,0], [0,1], [1,0], [1,1]]
2.  Y = [[0], [1], [1], [0]]
```

Figura 37. Patrones de entrenamiento para XOR

Creación del modelo

Para el entrenamiento de este problema se va a utilizar un modelo secuencial y constará de la capa de entrada de dimensión 2, una capa oculta de 8 neuronas con función de activación “tanh” y una capa de salida de dimensión 1 con función de activación “sigmoid”.

```
1.  model = Sequential()
2.  model.add(Dense(8, input_dim=2,activation='tanh'))
3.  model.add(Dense(1,activation='sigmoid')
```

Figura 38. Creación del modelo XOR

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8)                 24
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

Figura 39. Resumen red neuronal XOR

Compilación de modelo

```
1.  model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

Figura 40. Compilación de red neuronal XOR

Entrenamiento del modelo

Para el entrenamiento de la red neuronal por la característica que tiene tanto la función de activación sigmoid y tanh de que la convergencia es lenta, para este ejemplo se realizara el entrenamiento con 1000 épocas.

```
1.  history = model.fit(X, Y, epochs=1000)
```

Figura 41. Entrenamiento de red neuronal XOR

Resultados

La manera en que se comprueba la eficiencia de la red entrenada para problema XOR, es comparando la salida deseada con la salida estimada, esto se logra a través del siguiente código:

```
1.  print(model.predict_proba(X))
```

```
[[0.02926576]
 [0.9370866 ]
 [0.93999803]
 [0.07638142]]
```

Figura 42. Predicción de la red neuronal XOR usando todas las combinaciones posibles.

Tabla 4
Comparación de salida deseada y estimada de red neuronal XOR

X	Y	Salida Deseada	Salida Estimada
0	0	0	0.02926576
0	1	1	0.9370866
1	0	1	0.93999803
1	1	0	0.07638142

Para poder visualizar de una mejor manera se emplea el siguiente código:

```
print ('Salida Deseada:  ', [i[0] > 0.9 for i in Y])
print ('Salida Predicha: ', [i[0] > 0.9 for i in model.predict(X)])
```

```
Salida Deseada:   [False, True, True, False]
Salida Predicha:  [False, True, True, False]
```

Figura 43. Resultados RN XOR

Podemos observar que efectivamente, la red cumple con el objetivo y es capaz de solucionar el problema XOR.

3.1.2. Identificación de funciones

Normalmente en la industria se encuentran procesos que pueden llegar a ser representados por medio de una ecuación o función dinámica, esto nos puede servir para tareas de diagnóstico de fallos, para diseñar controladores, etc. En este ejemplo se realizará la identificación de una función sencilla como es $z = x^2 + y^2$.

Elaboración de la red neuronal

Patrones de entrenamiento

Para la obtención de los patrones se realizará un arreglo para la coordenada x y un arreglo para la coordenada y que estarán en los intervalos: $-1 \geq x \geq 1$ & $-1 \geq y \geq 1$. Para obtener todas las combinaciones posibles entre x e y, se hará uso del comando meshgrid de la librería NumPy, como se muestra a continuación:

```
x = np.linspace(-1,1,100)  ; y = np.linspace(-1,1,100)
x,y = np.meshgrid(x,y)
#Funcion a identificar
z = x**2+y**2
#Valores de entrada a la red neuronal
U=np.array([x,y]).T.reshape(-1,2)
#Valores de salida (targets) a la red neuronal
Z1=np.array([z]).T.reshape(-1,1)
```

Figura 44. Patrones de entrenamiento – Identificación de funciones

Creación del modelo

Para el entrenamiento de este problema se va a utilizar un modelo secuencial y constará de la capa de entrada de dimensión 2 (x,y), una capa oculta de 32 neuronas con función de activación “relu” y una capa de salida de dimensión 1 (z) con función de activación “relu”.

```
1.  model = Sequential()
2.  model.add(Dense(32, input_dim=2,activation='relu'))
3.  model.add(Dense(1,activation='relu')
```

Figura 45. Creación del modelo – Identificación de funciones

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8)                 24
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

Figura 46. Resumen red neuronal – Identificación de funciones

Compilación de modelo

```
1.  model.compile(loss='mean_square_error, optimizer='Adam', metrics=['accuracy'])
```

Figura 47. Compilación de red neuronal – Identificación de funciones

Entrenamiento del modelo

Para el entrenamiento de la red neuronal por la característica que tiene tanto la función de activación sigmoid y tanh de que la convergencia es lenta, para este ejemplo se realizara el entrenamiento con 1000 épocas.

```
1.  history = model.fit(U, Z1, epochs=500)
```

Figura 48. Entrenamiento de red neuronal – Identificación de funciones

Resultados

Para poder observar de mejor manera la eficiencia de la red neuronal, se grafica la función en tres dimensiones tanto de la función original como de la identificación por medio de la red.

```
1.  import numpy as np
2.  from matplotlib import pyplot as plt
3.  from mpl_toolkits.mplot3d import Axes3D
4.
5.  fig = plt.figure()
6.  ax = fig.add_subplot(111, projection='3d')
7.  #label axes
8.  ax.set_xlabel('x')
9.  ax.set_ylabel('y')
10. ax.set_zlabel('z')
11. #Graficar
12. ax.plot_surface(x,y,z,linewidth=0, antialiased=False, shade = True, alpha = 0.5)

13. plt.show()
```

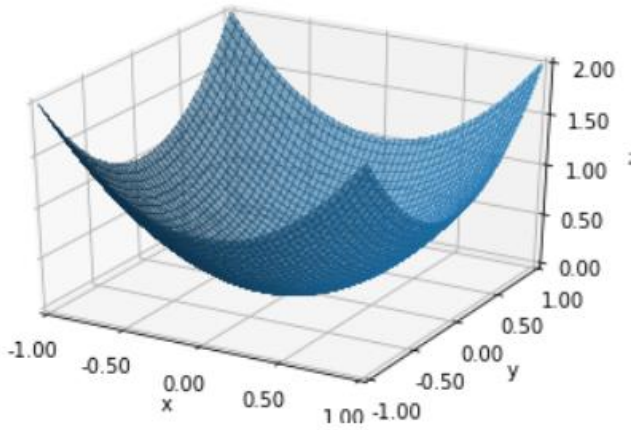


Figura 49. Grafica de la función $z = x^2 + y^2$

Ahora se presenta la gráfica de la identificación de la función.

```
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
#Se optiene la z estimada, introduciendo todos los valores de entrada U
Z11=np.array([model.predict(U)]).T.reshape(-1,100)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
#label axes
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
#Graficar
ax.plot_surface(x,y,Z11,linewidth=0, antialiased=False, shade = True, alpha = 0.5)
plt.show()
```

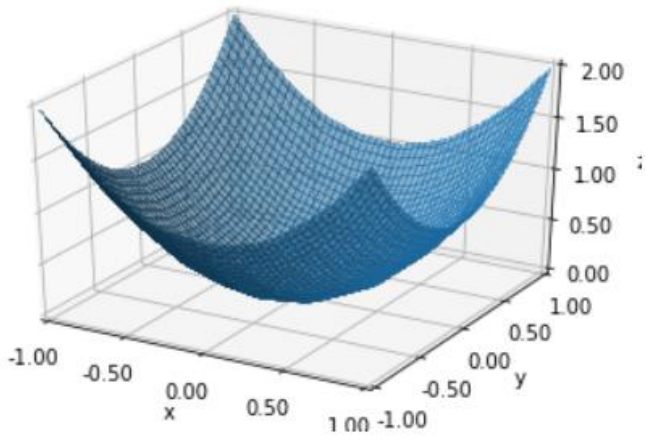


Figura 50. Grafica de la función identificada $z = x^2 + y^2$

Podemos observar que efectivamente, la red cumple con el objetivo y es capaz de realizar la identificación de funciones.

Se evalúa con MSE, lo cual se obtiene:

```
import sklearn
from sklearn.metrics import mean_squared_error
from sklearn.metrics import log_loss
ypred= model.predict(U)
print('ECM =',(mean_squared_error(Z1,ypred)))
```

```
ECM = 0.00011054337602910533
```

Se puede observar que el error entre el modelo predicho y la función real es menor al 1%

3.2. Ejemplos de Aplicación de Identificación

A continuación, se exponen problemas de identificación de sistemas dinamicos (Antena con péndulo invertido, Tanques Acoplados y Viga y Bola).

3.2.1. Identificación del sistema Antena con péndulo invertido

Se presenta el siguiente sistema de péndulo invertido

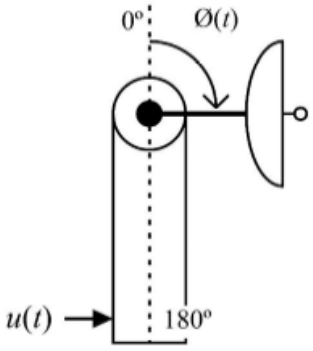


Figura 51. Antena con péndulo invertido

El brazo del ángulo de la antena ϕ es controlado aplicando una corriente al motor dc el cual está unido al péndulo. Este sistema puede ser representado por las siguientes ecuaciones.

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ 9.81 * \sin(x_1) - 2 * x_2 + u \end{bmatrix}$$

Donde,

$$x_1 = \phi$$

$$x_2 = \frac{d\phi}{dt}$$

Y u, es la fuerza aplicada al péndulo por el motor.

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para identificación del sistema.

Para obtener los ejemplos de entrenamiento, primero trasladamos la ecuación de movimiento a diagrama de bloques:

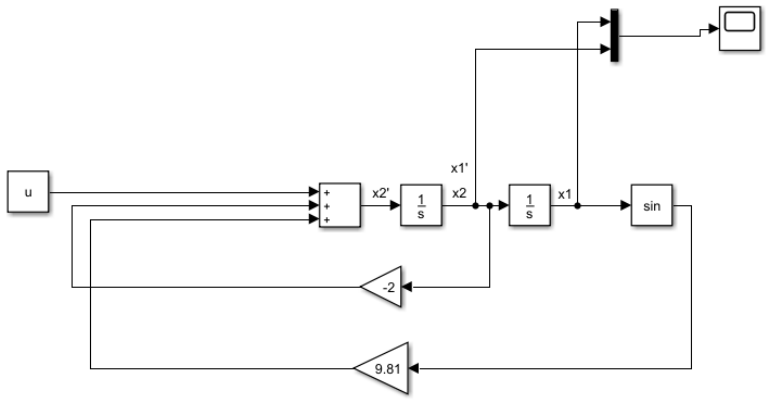


Figura 52. Diagrama de bloques Antena con péndulo invertido

Luego agregamos bloques para obtener las condiciones finales, y posterior obtener la variación de las variables de estado.

El procedimiento es obtener una malla de valores de las entradas (ángulo, velocidad, fuerza), posteriormente simular el sistema para cada punto de dicha malla y de esta forma obtener las variaciones de los estados (variación ángulo, variación velocidad) luego de un tiempo de discretización Ts=50ms.

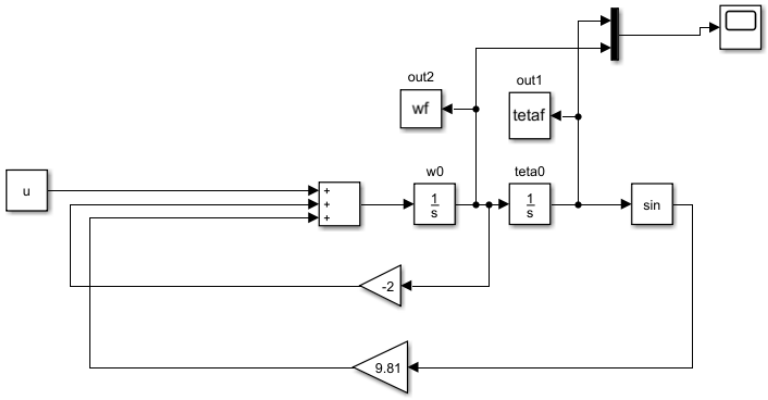


Figura 53. Diagrama de Antena para obtención de variación de variables de estado

```
%Script para obtencion de ejemplos de entrenamiento Antena
%Identificacion

timestep=50e-3

ang=[-20:22:200]*pi/180; %Posibles angulos
vel=[-90:36:90]*pi/180; %Posibles velocidades
u=[-30:6:30]; %Señal de control (Fuerza)
angle2 = [-20:10:200]*pi/180;

%Para tener todas las combinaciones posibles de los valores
%ya definidos.
Pm = combvec(ang,vel,u)

%Obtencion de variacion de estados de salida.
Tm=[]
for i=1:length(Pm)
    i
    u=Pm(3,i);
    w0=Pm(2,i);
    teta0=Pm(1,i);
    [t,x]=sim('identlineal',[0 timestep]);
    dteta=tetaf-teta0;  %variacion teta
    dw=wf-w0;   %variacion velocidad
Tm=[Tm [dteta ;dw]];

end

%Guardar en archivo de excel

filename = 'InIdenAntena.xlsx'; xlswrite(filename,Pm)
filename = 'OutIdenAntena.xlsx'; xlswrite(filename,Tm)
```

Figura 54. Script para adquisición de patrones de entrenamiento - Identificación Antena

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy.

```
#Se importa de los archivos exportados desde MATLAB
Pm = np.matrix(pd.read_csv("InIdenAntena.csv", header=None).values)
Tm = np.matrix(pd.read_csv("OutIdenAntena.csv", header=None).values)

#Se realiza la transpuesta
Pm= Pm.T.reshape(-1,3)
Tm= Tm.T.reshape(-1,2)

#Se separa de todos los patrones de entrenamiento, para poder hacer
#un test para ver el rendimiento de la red
X_train, X_test, y_train, y_test = train_test_split(Pm, Tm, test_size=0.20, random_state=40)
```

Figura 55. Patrones de entrenamiento - Identificación Antena

Creación de modelo

Para el entrenamiento de este problema se va a utilizar un modelo secuencial y constará de la capa de entrada de dimensión 3 (ángulo, velocidad, fuerza), dos capas ocultas de 8 neuronas con función de activación “relu” y una capa de salida de dimensión 2 (variación ángulo, variación velocidad).

```
model = Sequential()
model.add(Dense(8, input_dim=3,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(2))
```

Figura 56. Creación del modelo en Python – identificación Antena

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8)                 32
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 18
=================================================================
Total params: 122
Trainable params: 122
Non-trainable params: 0
_________________________________________________________________
```

Figura 57. Resumen red neuronal - identificación Antena

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 58. Compilación de red neuronal identificación – Antena

Entrenamiento del modelo

Para el entrenamiento de la red neuronal se ha considerado 700 épocas.

```
1.  history = model.fit(X_train, y_train, epochs=700)
```

Figura 59. Entrenamiento de red neuronal identificación - Antena

Prueba de rendimiento

En el entrenamiento de la red neuronal se ha usado el 80% de los patrones de entrenamiento, para hacer la prueba se usa el 20% restante y se calcula el error cuadrático medio entre la predicción del modelo con todas las combinaciones posibles de ese 80 % y la predicción del modelo con los restantes datos (20%).

```
1.  import sklearn
2.  from sklearn.metrics import mean_squared_error
3.  pred_train= model.predict(X_train)
4.  print('ECM 80% =',np.sqrt(mean_squared_error(y_train,pred_train)))
5.  pred= model.predict(X_test)
6.  print('ECM 20% =',np.sqrt(mean_squared_error(y_test,pred)))
```

```
ECM 80% = 0.010118983570733186
ECM 20% = 0.010421027022298019
```

Figura 60. Prueba de rendimiento RN identificación - Antena

Exportación del modelo

```
1.  #GUARDAR MODELO
2.  from keras.models import load_model
3.  model.save('IdentAntena.h5')
```

Figura 61. Exportación de red neuronal identificación - Antena

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “fcnide.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El archivo que contiene el modelo “IdentAntena.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = fcnide(teta0,w0,u)
modelfile = 'IdentAntena.h5';
net = importKerasNetwork(modelfile);
yu=predict(net,[teta0 w0 u]);
%Conversión de un vector simple a %valor
flotante
yu=double(yu)
y = yu;
```

Figura 62. Función en MATLAB para uso del modelo importado – Identificación Antena

Resultados

Se presenta el modelo exportado en imagen formato png.

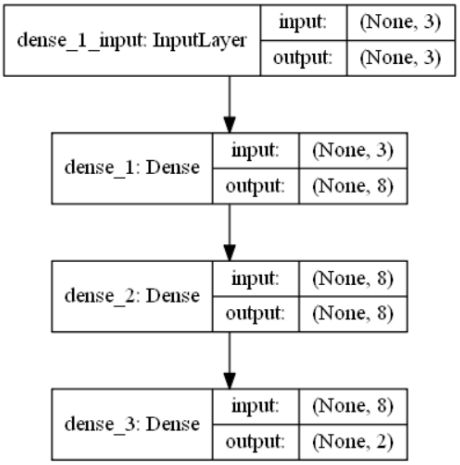


Figura 63. Modelo RN Antena

Se realiza la simulación en paralelo del sistema en diagrama de bloques y de la red neuronal.

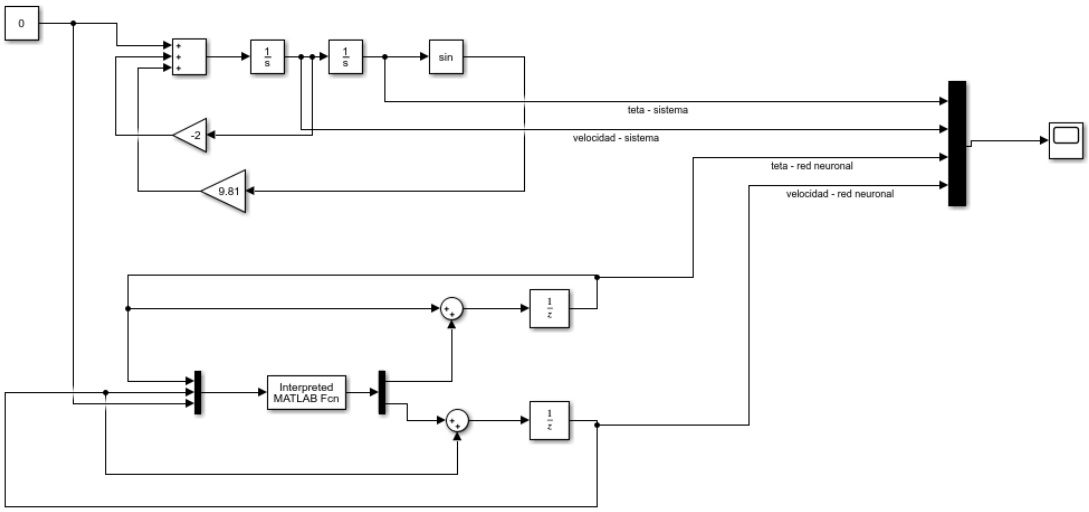


Figura 64. Sistema vs RN - Identificación Antena

Se muestra las curvas de cada una de las variables de estado, tanto del sistema (líneas continuas) y de la red neuronal (líneas punteadas).

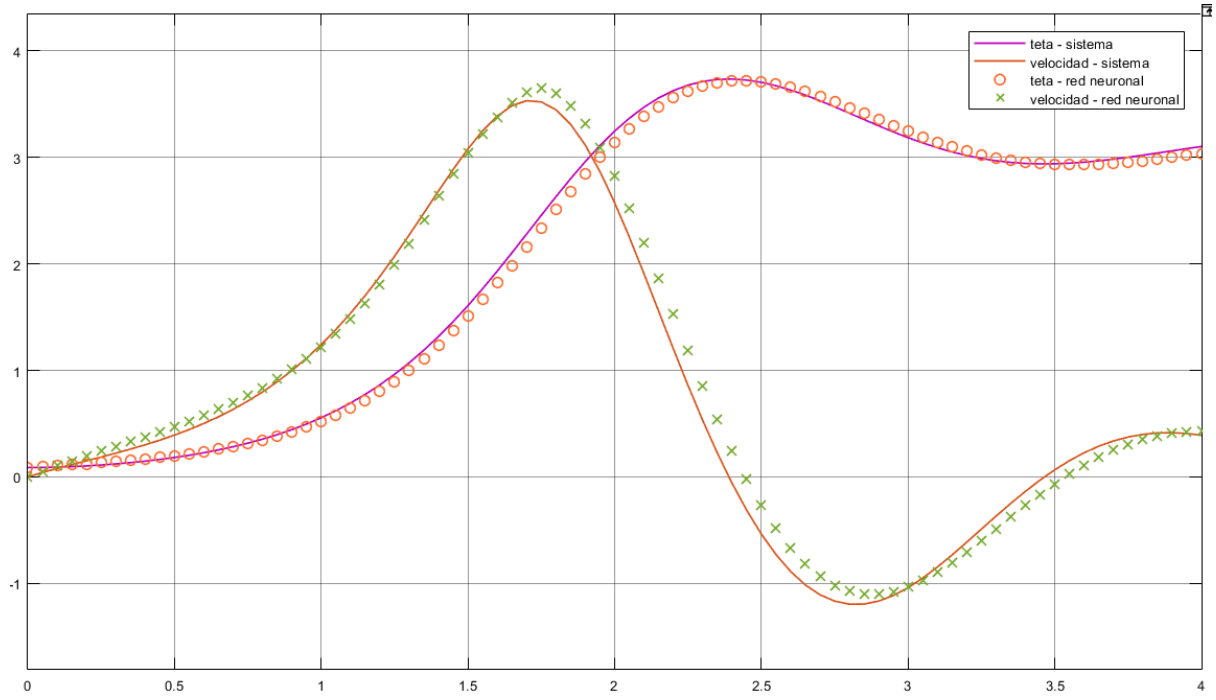


Figura 65. Curvas de variables de estado en lazo abierto – Antena

3.2.2. Identificación del sistema Tanques Acoplados

Se presenta el siguiente sistema dos tanques acoplados

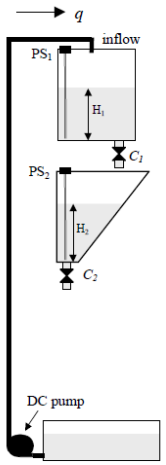


Figura 66. Esquema Tanques Acoplados

Se considera el modelo proporcionado por Inteco[1]. Las simulaciones de los tanques que se usará para las simulación y adquisición de datos se presenta a continuación:

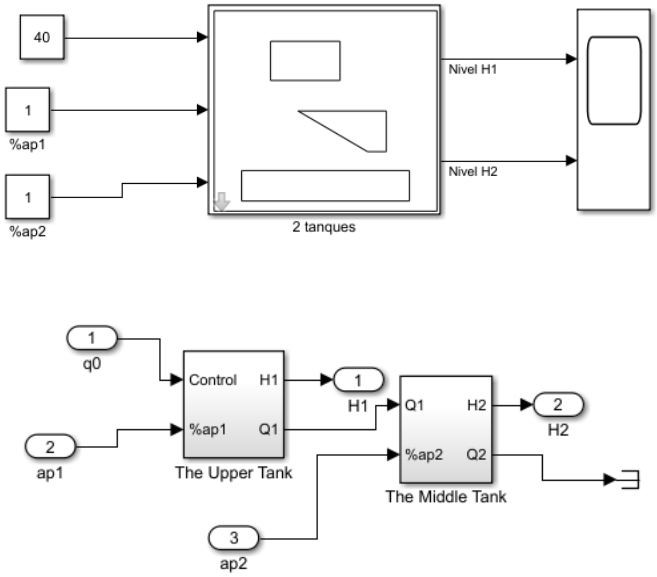


Figura 67. Esquema en Simulink de 2 tanques acoplados.

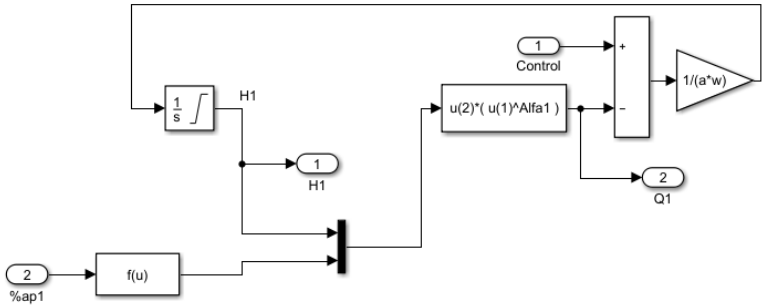


Figura 68. Esquema en Simulink tanque superior.

[1] Inteco es el fabricante de los equipos de laboratorio de servomecanismos adquirido por la Universidad de las Fuerzas Armadas – ESPE.

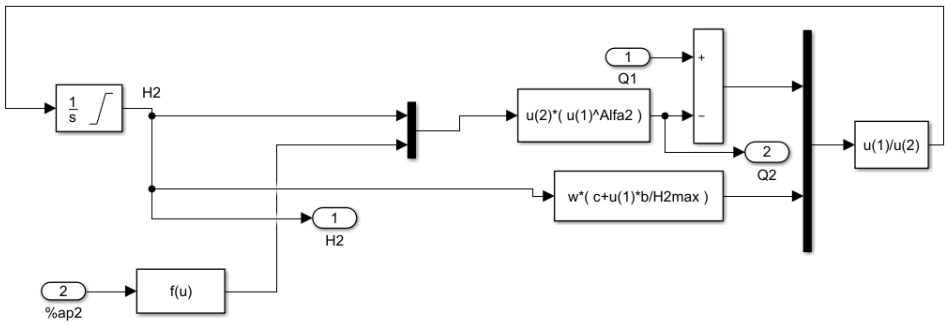


Figura 69. Esquema en Simulink tanque inferior.

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para identificación del sistema.

Se agrega bloques de entrada y salida para obtener las condiciones finales, y posterior obtener la variación de las variables de estado.

El procedimiento es obtener una malla de valores de las entradas (altura tanque 1, apertura válvula 1, altura tanque 2, apertura válvula 2, caudal), posteriormente simular el sistema para cada punto de dicha malla y de esta forma obtener las variaciones de los estados (variación altura tanque 1, variación altura tanque 2) luego de un tiempo de discretización Ts=1 seg.

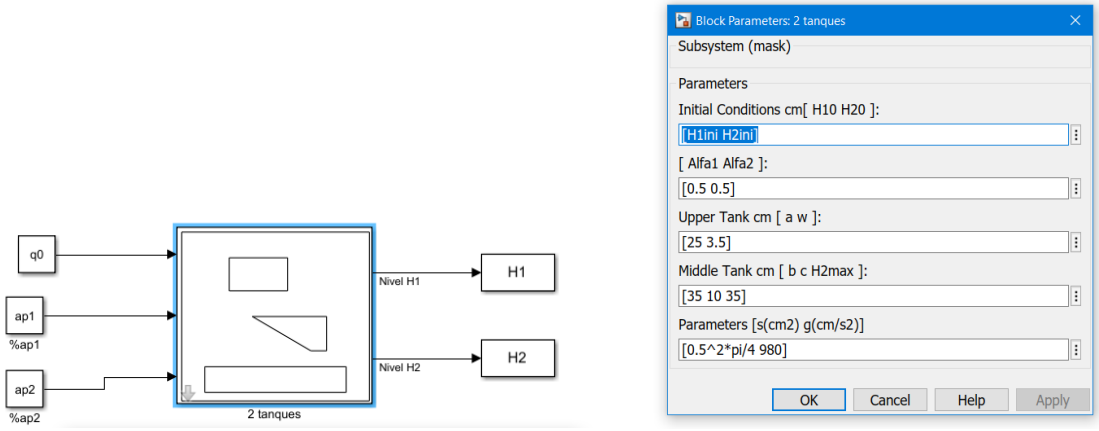


Figura 70. Diagrama de Tanque Acoplados para obtención de variación de variables de estado

Se ejecuta el siguiente Script que simulará el sistema para cada combinación posible de las variables de estado y la señal de control.

```
%Script para obtencion de ejemplos de entrenamiento Tanques Acoplados
%Identificacion en MATLAB

H1iniV=[1:5:25];%Posibles alturas tanque 2
H2iniV=[1:5:25];%Posibles alturas tanque 2
ap1V=[0.1:0.1:1]; %apertura de valvula 1
ap2V=[0.1:0.1:1]; %apertura de valvula 2
q0V=[10:10:40]; %señal de control - caudal
%Combinacion de valores de vectores : combvec

Pm=combvec(H1iniV,H2iniV,q0V,ap1V,ap2V)

dh1V=[]
dh2V=[]
Tm=[]
for i=1:length(Pm)
    i
  H1ini=Pm(1,i);
  H2ini=Pm(2,i);
  q0=Pm(3,i);
  ap1=Pm(4,i);
  ap2=Pm(5,i);
  sim('two_tank_1',[0 50e-3]);
  dh1=H1-H1ini;
  dh2=H2-H2ini;
  Tm=[Tm [dh1 ;dh2]];
end

%Exportar los datos
filename = 'InIdenTanque.csv'; csvwrite(filename,Pm')
filename = 'OutIdenTanque.csv'; csvwrite(filename,Tm')
```

Figura 71. Adquisición de patrones de entrenamiento – Identificación Tanques Acoplados

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy.

```
1.  #Se importa de los archivos exportados desde MATLAB
2.  Pm = np.matrix(pd.read_csv("inIdentTanque.csv", header=None).values)
3.  Tm = np.matrix(pd.read_csv("OutIdenTanque.csv", header=None).values)
4.
5.  #Se separa de todos los patrones de entrenamiento, para poder hacer
6.  #un test para ver el rendimiento de la red
7.  X_train, X_test, y_train, y_test = train_test_split(Pm, Tm, test_size=0.20, random_state=40)
```

Figura 72. Patrones de entrenamiento identificación – Identificación Tanques Acoplados

Creación de modelo

Para el entrenamiento de este problema se va a utilizar un modelo secuencial y constará de la capa de entrada de dimensión 5, una capa oculta de 32 neuronas con función de activación “tanh” y una capa de salida de dimensión 2.

```
1.  model = Sequential()
2.  model.add(Dense(32, input_dim=5,activation='tanh'))
3.  model.add(Dense(2))
```

Figura 73. Creación del modelo en Python – Identificación Tanques Acoplados

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 32)                192
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 66
=================================================================
Total params: 258
Trainable params: 258
Non-trainable params: 0
```

Figura 74. Resumen red neuronal - Identificación Tanques Acoplados

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 75. Compilación de red neuronal – Identificación Tanques Acoplados

Entrenamiento del modelo

Para el entrenamiento de la red neuronal se ha considerado 1000 épocas.

```
1.  history = model.fit(X_train, y_train, epochs=1000)
```

Figura 76. Entrenamiento de red neuronal – Identificación Tanques Acoplados

Prueba de rendimiento

En el entrenamiento de la red neuronal se ha usado el 80% de los patrones de entrenamiento, para hacer la prueba se usa el 20% restante y se calcula el error cuadrático medio entre la predicción del modelo con todas las combinaciones posibles de ese 80 % y la predicción del modelo con los restantes datos (20%).

```
1.  import sklearn
2.  from sklearn.metrics import mean_squared_error
3.  pred_train= model.predict(X_train)
4.  print('ECM 80% =',np.sqrt(mean_squared_error(y_train,pred_train)))
5.  pred= model.predict(X_test)
6.  print('ECM 20% =',np.sqrt(mean_squared_error(y_test,pred)))
```

```
ECM 80% = 0.10043386866201451
ECM 20% = 0.09682708301788577
```

Figura 77. Prueba de rendimiento RN – Identificación Tanques Acoplados

Exportación del modelo

```
1.  #GUARDAR MODELO
2.  from keras.models import load_model
3.  model.save('my_modelTanque.h5')
```

Figura 78. Exportación de red neuronal – Identificación Tanques Acoplados

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “identank.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El archivo que contiene el modelo “IdentAntena.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = idetank(h1,v1,h2,v2,q)
modelfile = 'my_modelTanque.h5';
net = importKerasNetwork(modelfile);
yu=predict(net,[h1 v1 h2 v2 q]);
%Conversión de un vector simple a %valor
flotante
yu=double(yu);
y = yu;
```

Figura 79. Función en MATLAB – Identificación Tanques Acoplados

Resultados

Se presenta el modelo exportado en imagen formato png.

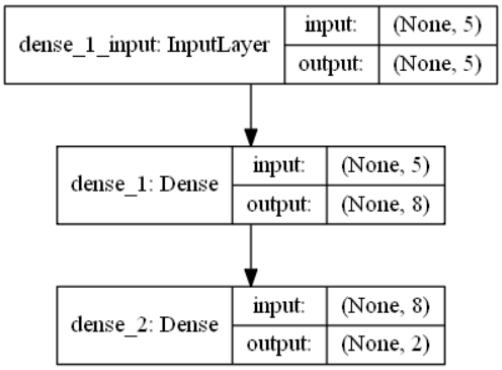


Figura 80. Modelo RN – Identificación Tanques Acoplados

Se realiza la simulación en paralelo del sistema en diagrama de bloques y de la red neuronal.

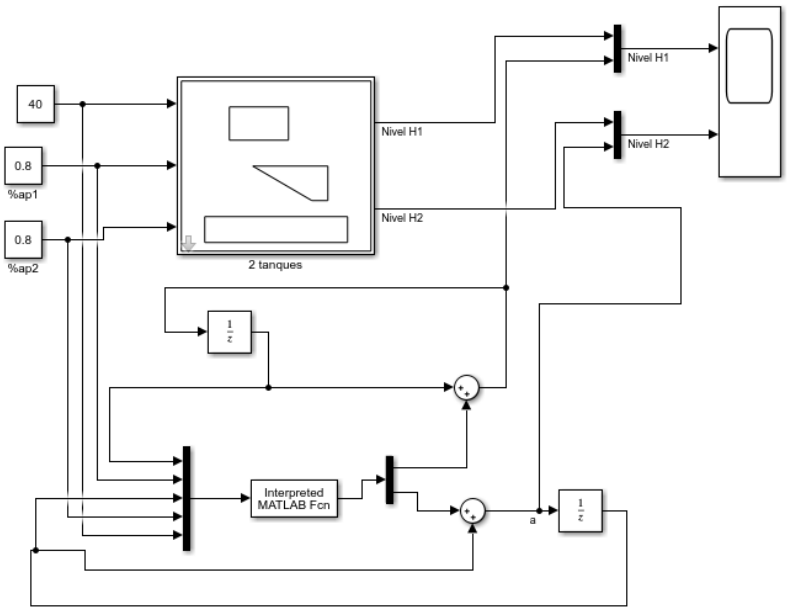


Figura 81. Sistema vs RN – Identificación Tanques Acoplados

Se muestra las curvas de cada una de las variables de estado, tanto del sistema (líneas continuas) y de la red neuronal (líneas punteadas).

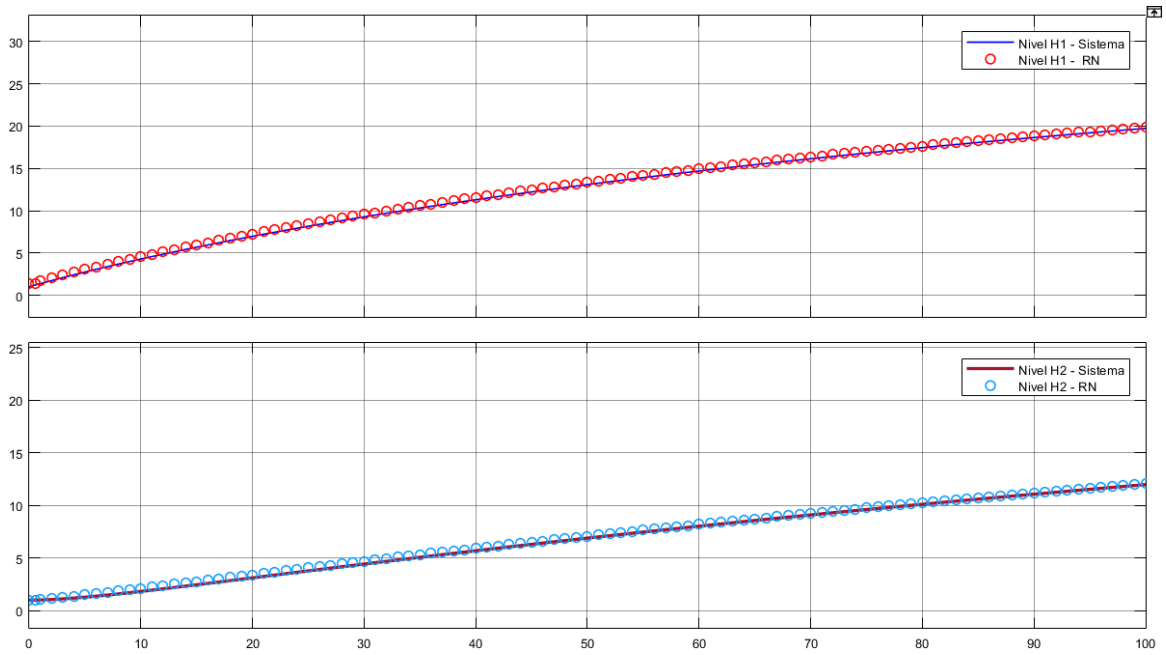


Figura 82. Curvas de variables de estado en lazo abierto – Identificación Tanques Acoplados

3.2.3. Identificación del sistema Viga y Bola

Se presenta el siguiente sistema de viga y bola

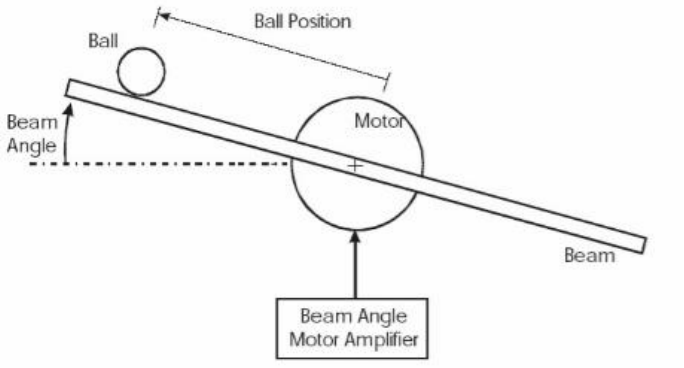


Figura 83. Diagrama de viga y bola

La viga es controlada aplicando una corriente al motor dc del cual está unido. Este sistema puede ser representado por las siguientes ecuaciones:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -0.6 * x_2 - 4 * x_1 + 22 * u \end{bmatrix}$$

Donde,

$x_1 = p$ (posición de la bola)

$x_2 = \frac{dp}{dt}$ (velocidad de la bola)

Y u, es la fuerza aplicada a la viga por el motor.

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para identificación del sistema.

Para obtener los ejemplos de entrenamiento, primero trasladamos la ecuación de movimiento a diagrama de bloques:

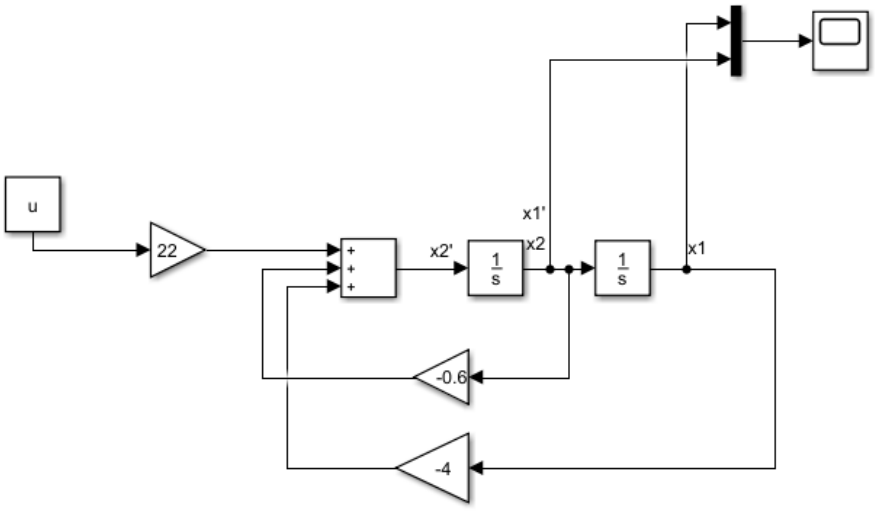


Figura 84. Diagrama de bloques - Viga y Bola

Luego agregamos bloques para obtener las condiciones finales, y posterior obtener la variación de las variables de estado.

El procedimiento es obtener una malla de valores de las entradas (posición, velocidad, fuerza), posteriormente simular el sistema para cada punto de dicha malla y de esta forma obtener las variaciones de los estados (variación posición, variación velocidad) luego de un tiempo de discretización Ts=50ms.

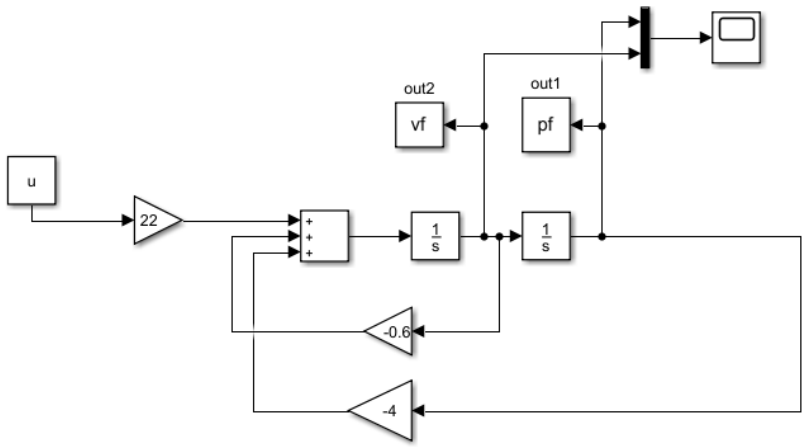


Figura 85. Diagrama para obtención de variación de variables de estado – Viga y Bola

```
%Identificacion de Viga y Bola
timestep=50e-3
%data de entrenamiento

ang=[-45:4:45]*pi/180; %señal de control
vel=[-0.7:0.1:0.7]; %posibles velocidades de bola
pos=0:0.05:1; %posibles posiciones de bola
pos2=0:0.03:1;
Pm = [combvec(pos,vel,ang) [pos2; zeros(2,length(pos2))]];

Tm=[]
for i=1:length(Pm)
    i
    u=Pm(3,i);
    v0=Pm(2,i);
    p0=Pm(1,i);
    sim('idenviga',[0 timestep]);
    dp=pf-p0;
dv=vf-v0;
Tm=[Tm [dp ;dv]];
end

%Exportar los datos
filename = 'InIdenViga.csv'; csvwrite(filename,Pm')
filename = 'OutIdenViga.csv'; csvwrite(filename,Tm')
```

Figura 86. Script para adquisición de patrones de entrenamiento Identificación – Viga y Bola

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy.

```
#Se importa de los archivos exportados desde MATLAB
Pm = np.matrix(pd.read_csv("InIdentViga.csv", header=None).values)
Tm = np.matrix(pd.read_csv("OutIdentViga.csv", header=None).values)

#Se hace un reshape
Pm= Pm.T.reshape(-1,3)
Tm= Tm.T.reshape(-1,2)

#Se separa de todos los patrones de entrenamiento, para poder hacer
#un test para ver el rendimiento de la red
X_train, X_test, y_train, y_test = train_test_split(Pm, Tm, test_size=0.20, random_state=40)
```

Figura 87. Patrones de entrenamiento – Viga y Bola

Creación de modelo

Para el entrenamiento de este problema se va a utilizar un modelo secuencial y constará de la capa de entrada de dimensión 3, dos capas ocultas de 8 neuronas con función de activación “relu” y una capa de salida de dimensión 2.

```
1.  model = Sequential()
2.  model.add(Dense(8, input_dim=3,activation='relu'))
3.  model.add(Dense(8,activation='relu'))
4.  model.add(Dense(2))
```

Figura 88. Creación del modelo en Python – Viga y Bola

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8)                 32
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 18
=================================================================
Total params: 122
Trainable params: 122
Non-trainable params: 0
_________________________________________________________________
```

Figura 89. Resumen red neuronal identificación – Viga y Bola

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 90. Compilación de red neuronal identificación – Viga y Bola

Entrenamiento del modelo

Para el entrenamiento de la red neuronal se ha considerado 300 épocas.

```
1.  history = model.fit(X_train, y_train, epochs=300)
```

Figura 91. Entrenamiento de red neuronal identificación – Viga y Bola

Prueba de rendimiento

En el entrenamiento de la red neuronal se ha usado el 80% de los patrones de entrenamiento, para hacer la prueba se usa el 20% restante y se calcula el error cuadrático medio entre la predicción del modelo con todas las combinaciones posibles de ese 80 % y la predicción del modelo con los restantes datos (20%).

```
1.  import sklearn
2.  from sklearn.metrics import mean_squared_error
3.  pred_train= model.predict(X_train)
4.  print('ECM 80% =',np.sqrt(mean_squared_error(y_train,pred_train)))
5.  pred= model.predict(X_test)
6.  print('ECM 20% =',np.sqrt(mean_squared_error(y_test,pred)))
```

```
ECM 80% = 0.0001358136942533623 5
ECM 20% = 0.0001599740936252843 7
```

Figura 92. Prueba de rendimiento RN identificación – Viga y Bola

Exportación del modelo

```
1.  #GUARDAR MODELO
2.  from keras.models import load_model
3.  model.save('my_modelViga.h5')
```

Figura 93. Exportación de red neuronal identificación – Viga y Bola

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “identviga.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El

archivo que contiene el modelo “my_modelViga.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = identviga(p,v,theta)
 modelfile = 'my_modelViga.h5';
 net = importKerasNetwork(modelfile);
 yu=predict(net,[p v theta]);
 yu=double(yu);
y = yu;
```

Figura 94. Función en MATLAB– Identificación Viga y Bola

Resultados

Se presenta el modelo exportado en imagen formato png.

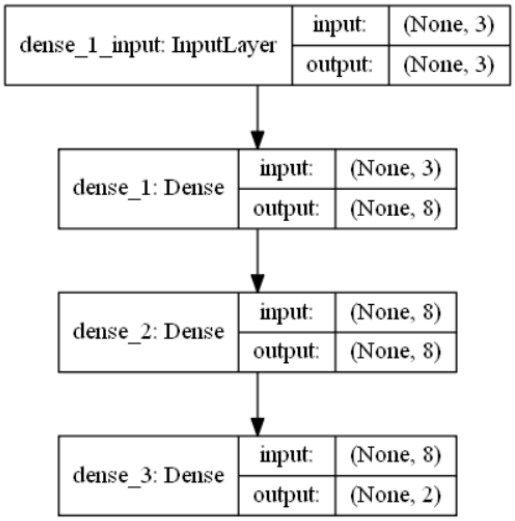


Figura 95. Modelo RN – Viga y Bola

Se realiza la simulación en paralelo del sistema en diagrama de bloques y de la red neuronal.

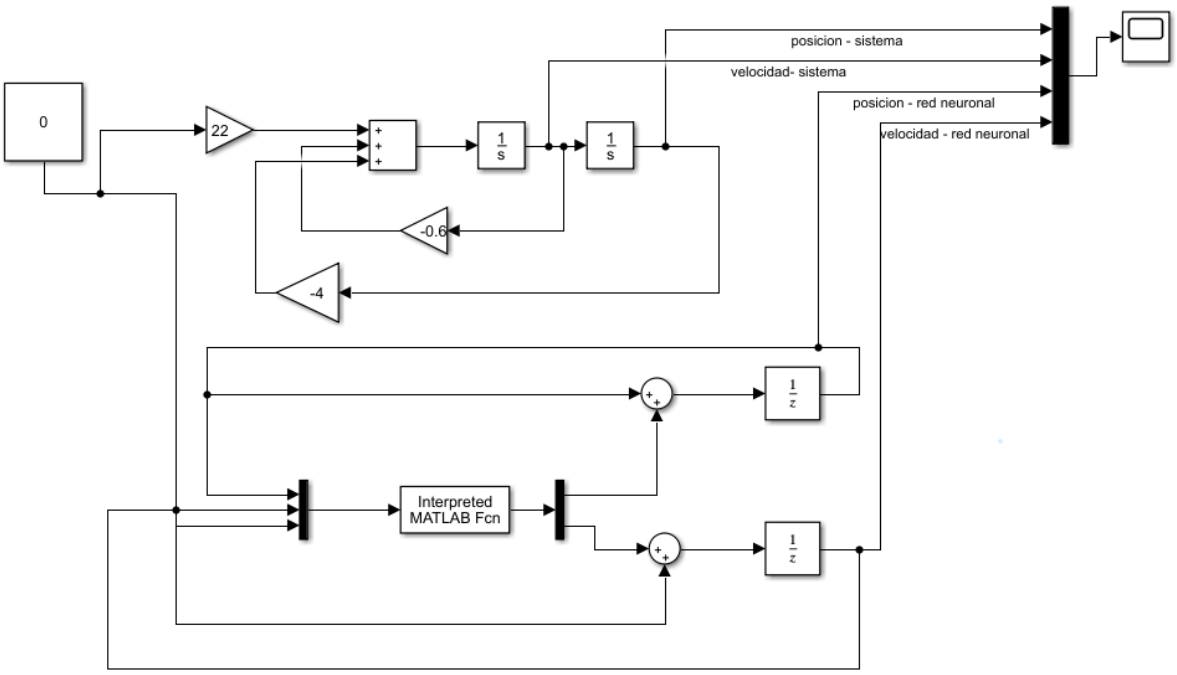


Figura 96. Sistema vs RN Identificación – Viga y Bola

Se muestra las curvas de cada una de las variables de estado, tanto del sistema (líneas continuas) y de la red neuronal (líneas punteadas).

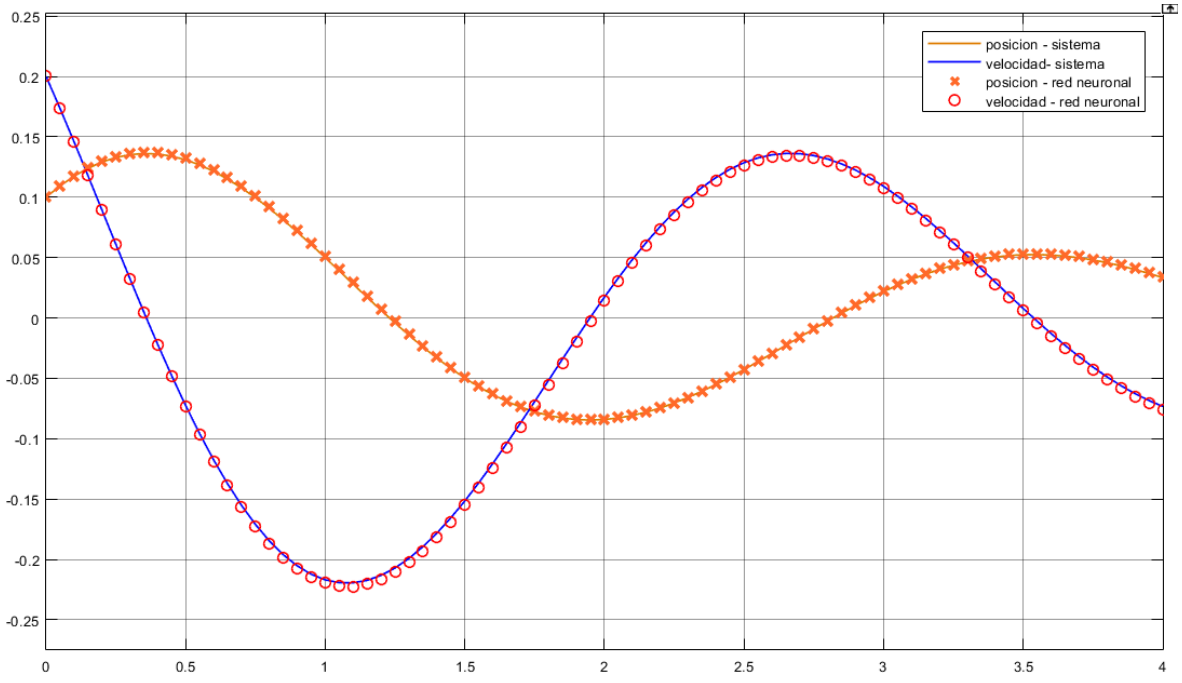


Figura 97. Curvas de variables de estado en lazo abierto – Viga y Bola

3.3. Ejemplos de aplicación de control con red neuronal inversa

3.3.1. Control con red neuronal inversa Antena con péndulo invertido

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para control neuronal inverso del sistema.

Ya obtenidos los patrones de entrenamiento para la identificación del sistema, para el entrenamiento del control neuronal inverso, se hará uso de los mismos, pero en este caso, las entradas serán las variables de estado y las variaciones de las de las variables de estado y su salida será la señal de control, como se explica en el apartado 2.5.2.

```
%Se considera que el workspace se tiene cargado los patrones de
% entrenamiento para la identificacion del sistema
Pm_inv= [Pm(1:2,:); Tm]
Tm_inv= [Pm(3,:)]
%Guardar en archivo de excel

filename = 'inControl.csv'; csvwrite(filename,Pm_inv)
filename = 'outControl.csv'; csvwrite(filename,Tm_inv)
```

Figura 98. Script para adquisición de patrones de entrenamiento – RN inversa Antena

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy.

```
1.  #Se importa de los archivos exportados desde MATLAB
2.  In = np.matrix(pd.read_csv("inControl.csv", header=None).values)
3.  Out = np.matrix(pd.read_csv("outControl.csv", header=None).values)
4.
5.  #Se realiza la transpuesta
6.  In= In.T.reshape(-1,4)
7.  Out= Out.T.reshape(-1,1)
```

Figura 99. Patrones de entrenamiento en Python – RN inversa Antena

Creación de modelo

Para el entrenamiento de este problema se va a utilizar un modelo secuencial y constará de la capa de entrada de dimensión 4, dos capas ocultas de 4 y 8 neuronas respectivamente, una capa con función de activación especial LeakyReLU y una capa de salida de dimensión 1.

```
1.  model = Sequential()
2.  model.add(Dense(4,input_dim=4))
3.  model.add(LeakyReLU(alpha=0.08))
4.  model.add(Dense(8))
5.  model.add(Dense(1))
```

Figura 100. Creación del modelo – RN inversa Antena

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 4)                 20
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 4)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 40
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 69
Trainable params: 69
Non-trainable params: 0
```

Figura 101. Resumen red neuronal – RN inversa Antena

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 102. Compilación de red neuronal – RN inversa Antena

Exportación del modelo

```
1.  #GUARDAR MODELO
2.  from keras.models import load_model
3.  model.save('my_modelControl.h5')
```

Figura 103. Exportación de red neuronal – RN inversa Antena

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “nnc.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El archivo que contiene el modelo “my_modeControl.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = nnc(teta0,w0,vteta0,vw0)
modelfile = 'my_modelControl.h5';
net = importKerasNetwork(modelfile);
yu=predict(net,[teta0 w0 vteta0 vw0]);
yu=double(yu);
y = yu;
```

Figura 104. Función en MATLAB para uso del modelo importado – RN inversa Antena

Resultados

Se presenta el modelo exportado en imagen formato png.

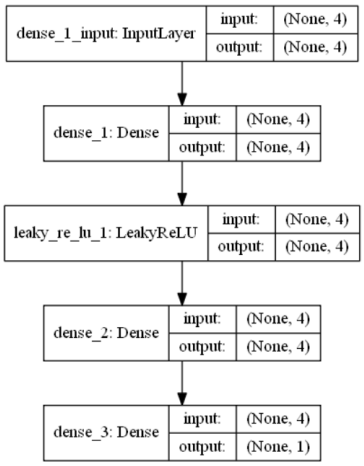


Figura 105. Modelo – RN inversa Antena

Se realiza la simulación del sistema siendo controlado por la red neuronal inversa.

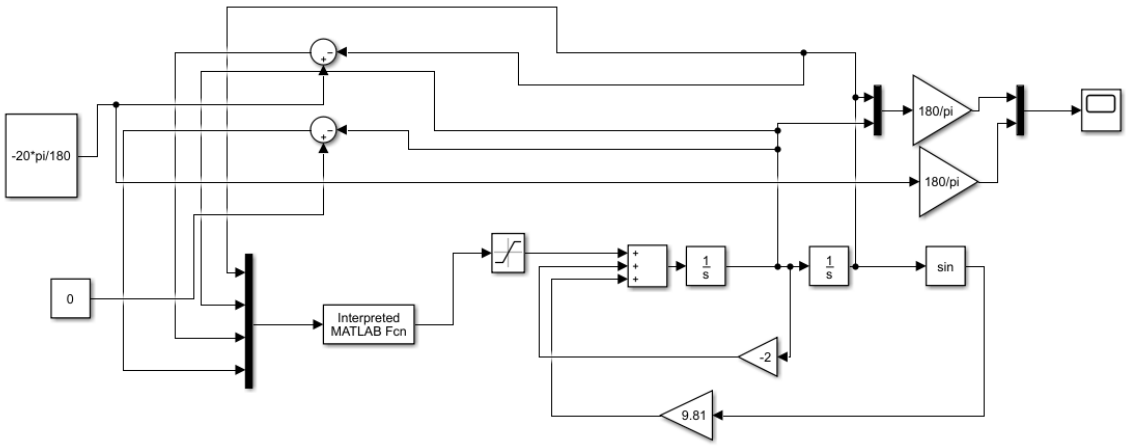


Figura 106. Sistema controlado por RN inversa – Antena

Se propone un SetPoint de -20 grados, con condiciones iniciales de $\theta = 60$ grados y velocidad angular $\omega = 20$ (grados/seg).

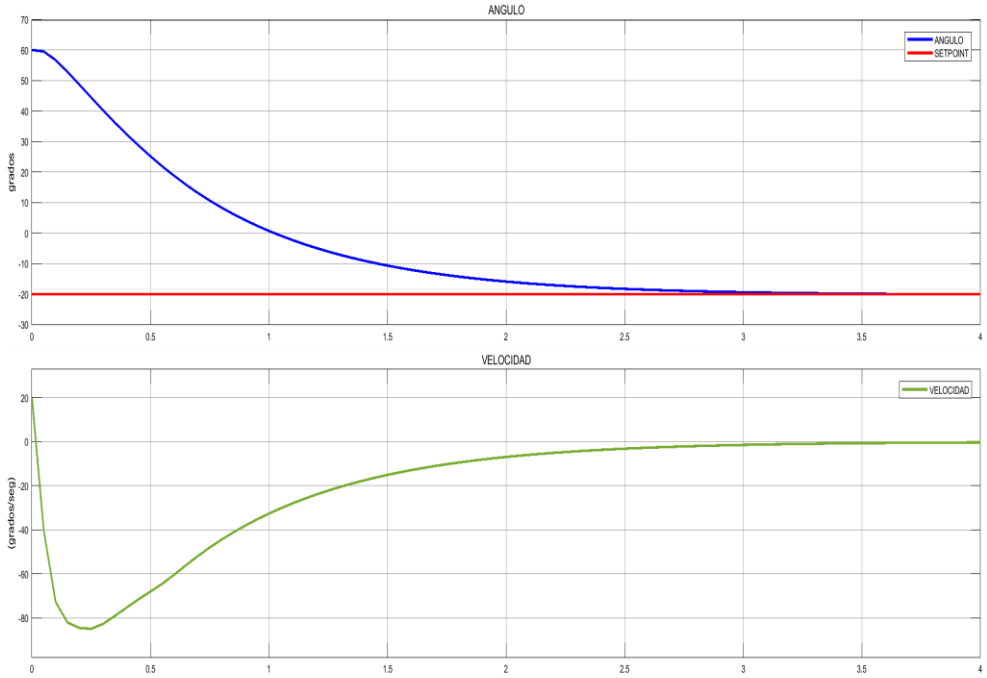


Figura 107. Respuesta de Control RN inversa – Antena

3.3.2. Control con red neuronal inversa Tanques Acoplados

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para control neuronal inverso del sistema.

Ya obtenidos los patrones de entrenamiento para la identificación del sistema, para el entrenamiento del control neuronal inverso, se hará uso de los mismos, pero en este caso, las entradas serán las variables de estado y las variaciones de las de las variables de estado y su salida será la señal de control, como se explica en el apartado 2.5.2.

```
%Se considera que el workspace se tiene cargado los patrones de
% entrenamiento para la identificacion del sistema
Pm_inv= [Pm(1,:);Tm(1,:);Pm(3,:);Tm(2,:);Pm(5,:)]
Tm_inv= [Pm(2,:);Pm(4,:)]
%Guardar en archivo de excel

filename = 'InContTanque.csv'; csvwrite(filename,Pm_inv)
filename = 'OutContTanque.csv'; csvwrite(filename,Tm_inv)
```

Figura 108. Script para adquisición de patrones de entrenamiento – RN inversa Tanques

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy.

```
#Se importa de los archivos exportados desde MATLAB
Pm = np.matrix(pd.read_csv("InControlTanque.csv", header=None).values)
Tm = np.matrix(pd.read_csv("outControlTanque.csv", header=None).values)
```

Figura 109. Patrones de entrenamiento en Python – RN inversa Tanques

Creación de modelo

Para el entrenamiento de este problema se va a utilizar un modelo secuencial y constará de la capa de entrada de dimensión 5, una capa oculta de 32 neuronas con función de activación “tanh” y una capa de salida de dimensión 2.

```
model = Sequential()
model.add(Dense(32, input_dim=5,activation='tanh'))
model.add(Dense(2))
```

Figura 110. Creación del modelo – RN inversa Tanques

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 32)                192
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 66
=================================================================
Total params: 258
Trainable params: 258
Non-trainable params: 0
_________________________________________________________________
```

Figura 111. Resumen red neuronal – RN inversa Tanques

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 112. Compilación de red neuronal – RN inversa Tanques

Exportación del modelo

```
1.  #GUARDAR MODELO
2.  from keras.models import load_model
3.  model.save('Control_Tanque.h5')
```

Figura 113. Exportación de red neuronal – RN inversa Tanques

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “contank.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El archivo que contiene el modelo “Control_Tanque.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = contank(h1,v1,h2,v2,q)
 modelfile = 'Control_Tanque.h5';
 net = importKerasNetwork(modelfile);
yu=predict(net,[h1 v1 h2 v2 q]);
yu=double(yu);
y = yu;
```

Figura 114. Función en MATLAB para uso del modelo importado – RN inversa Tanques

Resultados

Se presenta el modelo exportado en imagen formato png.

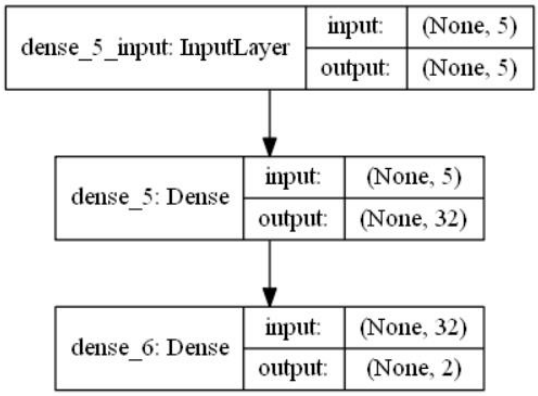


Figura 115. Modelo – RN inversa Tanques

Se realiza la simulación del sistema siendo controlado por la red neuronal inversa.

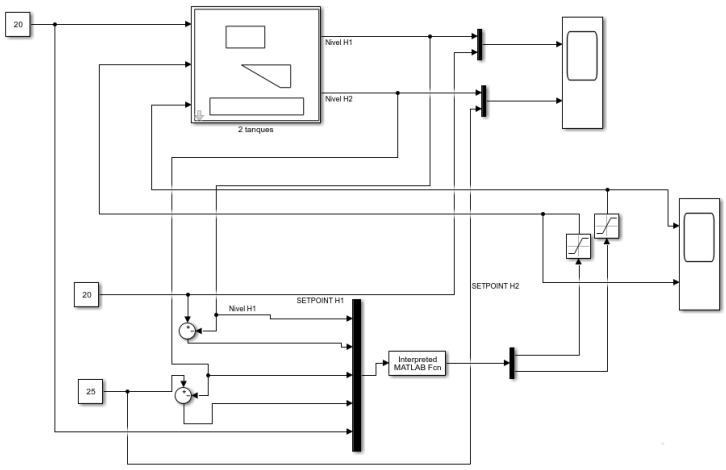


Figura 116. Sistema controlado por RN inversa – Tanques

Se propone un SetPoint para la altura del tanque 1 de 20 cm y de 25 cm para tanque 2, con condiciones iniciales, altura tanque 1 de 3 cm y altura tanque 2 de 5 cm y caudal $q = 20\ \left(\frac{m^3}{s}\right)$

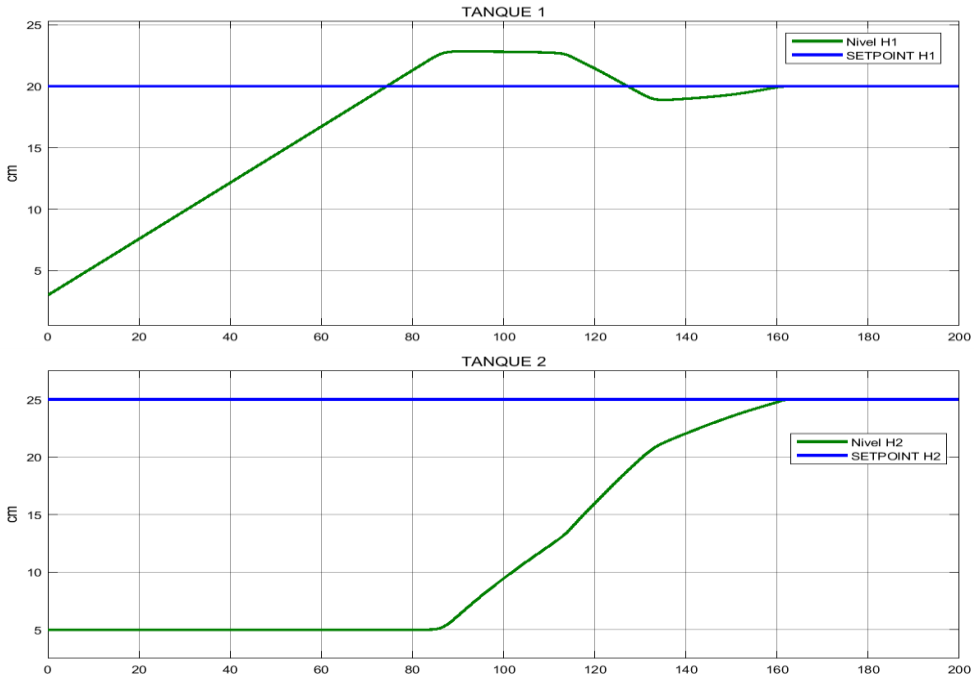


Figura 117. Respuesta de Control RN inversa - Tanques

3.3.3. Control con red neuronal inversa Viga y Bola

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para control neuronal inverso del sistema.

Ya obtenidos los patrones de entrenamiento para la identificación del sistema, para el entrenamiento del control neuronal inverso, se hará uso de los mismos, pero en este caso, las entradas serán las variables de estado y las variaciones de las de las variables de estado y su salida será la señal de control, como se explica en el apartado 2.5.2.

```
%Se considera que el workspace se tiene cargado los patrones de
% entrenamiento para la identificacion del sistema
Pm_inv= [Pm(1:2,:); Tm]
Tm_inv= [Pm(3,:)]
%Guardar en archivo de excel

filename = 'InContViga.csv'; csvwrite(filename,Pm_inv)
filename = 'OutContViga.csv'; csvwrite(filename,Tm_inv)
```

Figura 118. Script para adquisición de patrones de entrenamiento – RN inversa Viga

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy.

```
#Se importa de los archivos exportados desde MATLAB
Pm = np.matrix(pd.read_csv("InContViga.csv", header=None).values)
Tm = np.matrix(pd.read_csv("OutContViga.csv", header=None).values)

#Se realiza la transpuesta
Pm= Pm.T.reshape(-1,4)
Tm= Tm.T.reshape(-1,1)

#Se separa de todos los patrones de entrenamiento, para poder hacer
#un test para ver el rendimiento de la red
X_train, X_test, y_train, y_test = train_test_split(Pm, Tm, test_size=0.20, random_state=40)
```

Figura 119. Patrones de entrenamiento en Python – RN inversa Viga y Bola

Creación de modelo

Para el entrenamiento de este problema se va a utilizar un modelo secuencial y constará de la capa de entrada de dimensión 3, dos capas ocultas de 8 neuronas con función de activación “relu” y una capa de salida de dimensión 1.

```
model = Sequential()
model.add(Dense(4,input_dim=4))
model.add(LeakyReLU(alpha=0.08))
model.add(Dense(8))
model.add(Dense(1))
```

Figura 120. Creación del modelo – RN inversa Viga y Bola

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 4)                 20
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 4)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 40
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 69
Trainable params: 69
Non-trainable params: 0
```

Figura 121. Resumen red neuronal – RN inversa Viga y Bola

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 122. Compilación de red neuronal – RN inversa Viga y Bola

Entrenamiento del modelo

Para el entrenamiento de la red neuronal se ha considerado 1000 épocas.

```
1.  history = model.fit(X_train, y_train, epochs=1000)
```

Figura 123. Entrenamiento de red neuronal – RN inversa Viga y Bola

Prueba de rendimiento

En el entrenamiento de la red neuronal se ha usado el 80% de los patrones de entrenamiento, para hacer la prueba se usa el 20% restante y se calcula el error cuadrático medio entre la predicción del modelo con todas las combinaciones posibles de ese 80 % y la predicción del modelo con los restantes datos (20%).

```
1.  import sklearn
2.  from sklearn.metrics import mean_squared_error
3.  pred_train= model.predict(X_train)
4.  print('ECM 80% =',np.sqrt(mean_squared_error(y_train,pred_train)))
5.  pred= model.predict(X_test)
6.  print('ECM 20% =',np.sqrt(mean_squared_error(y_test,pred)))
```

```
ECM 80% =  0.0007137240570286509
ECM 20% =  0.0006982726734400875
```

Figura 124. Prueba de rendimiento – RN inversa Viga y Bola

Exportación del modelo

Para la exportación del modelo se utiliza el siguiente código:

```
1.  #GUARDAR MODELO
2.  from keras.models import load_model
3.  model.save('ControlViga.h5')
```

Figura 125. Exportación de red neuronal – RN inversa Viga y Bola

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “Contviga.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El archivo que contiene el modelo “ControlViga.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = Contviga(p,v,dp,dv)
 modelfile = 'ControlViga.h5';
net = importKerasNetwork(modelfile);
yu=predict(net,[p v dp dv]);
yu=double(yu)
y = yu;
```

Figura 126. Función en MATLAB para uso del modelo importado – RN inversa Viga y Bola

Resultados

Se presenta el modelo exportado en imagen formato png.

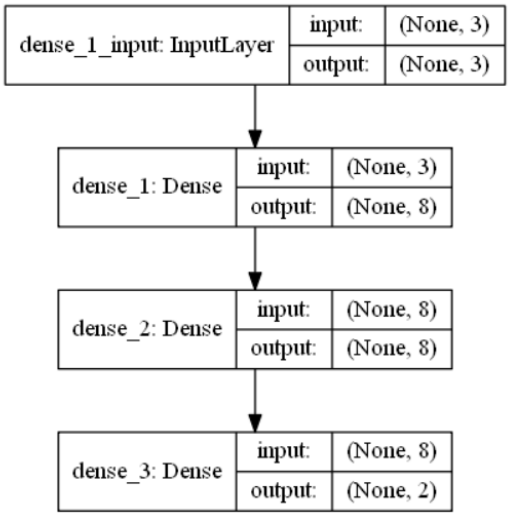


Figura 127. Modelo – RN inversa Viga y Bola

Se realiza la simulación del sistema siendo controlado por la red neuronal inversa.

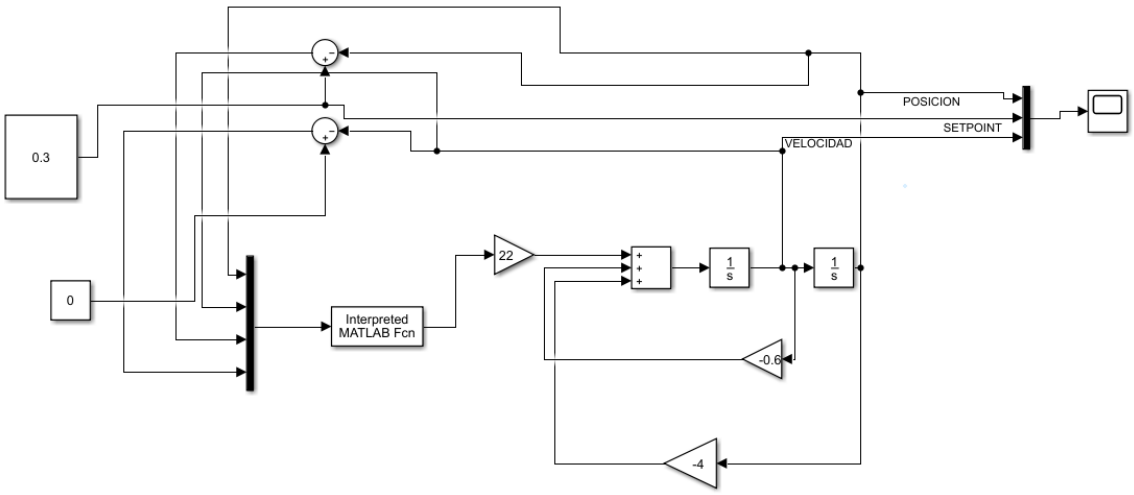


Figura 128. Sistema controlado por RN inversa Viga y Bola

Se propone un SetPoint de 30 cm (0.3m) , con condiciones de posición inicial $p = 0.8$ m y velocidad inicial $v = 0$ (m/seg).

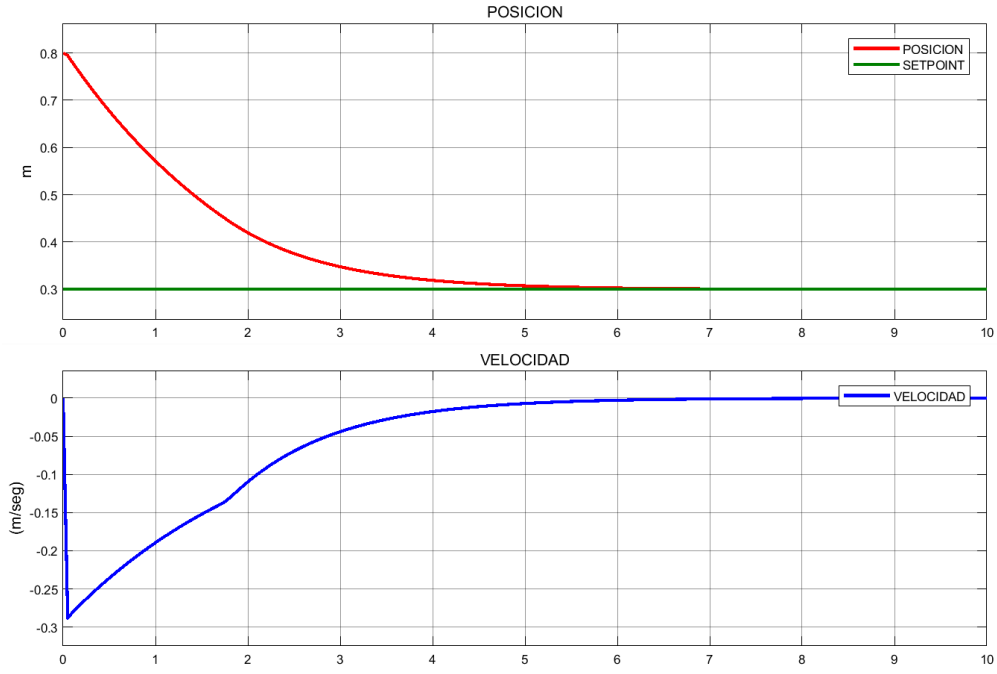


Figura 129. Curvas de variables de estado en lazo abierto – RN inversa Viga y Bola

3.4. Ejemplos de aplicación de control por Modelo de Referencia

3.4.1. Control con modelo de referencia para el sistema Antena

Se supone que se desea que el sistema en lazo cerrado responda a la dinámica dada por Modelo de Referencia Lineal, que se puede representar en las siguientes ecuaciones:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -9x_1 - 6x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 9r \end{bmatrix}$$

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para el controlador con modelo de referencia lineal

Para obtener los ejemplos de entrenamiento, primero trasladamos la ecuación de modelo de referencia lineal a diagrama de bloques:

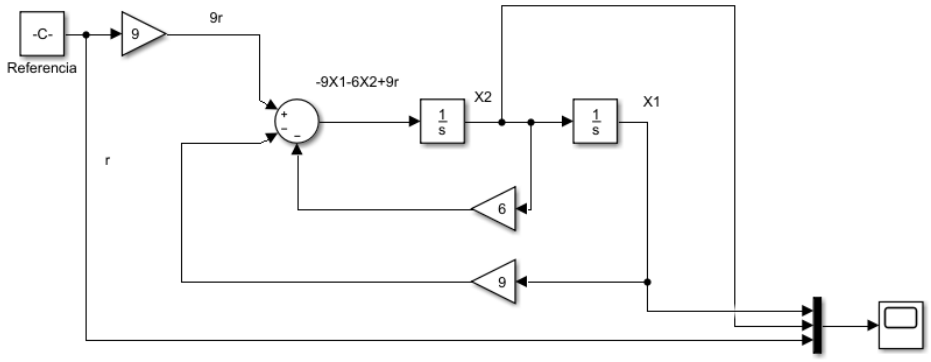
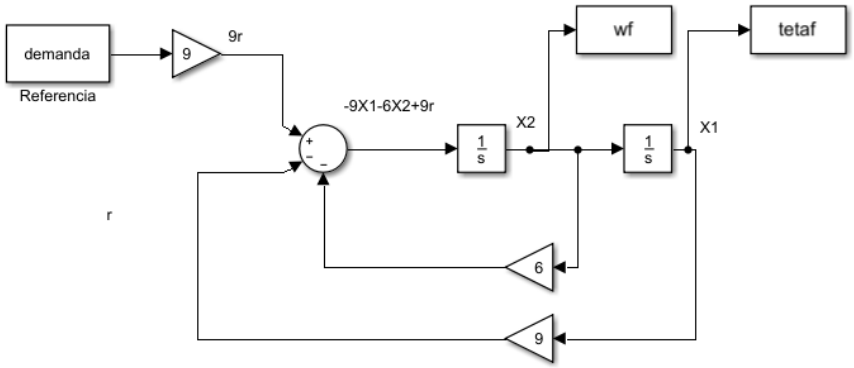


Figura 130. Diagrama de bloques – MRL Antena

Luego agregamos bloques para obtener las condiciones finales, y posterior obtener la variación de las variables de estado.




```
deg2rad=pi/180
angulo = [-10:10:190]*deg2rad;
velocidad= [-90:15:90]*deg2rad;
demanda = [-180:15:180]*deg2rad;
angle2 = [-10:10:190]*deg2rad;
Pc = [combvec(angulo,velocidad,demanda) [angle2; zeros(size(angle2)); angle2]];
Tc=[]
for i=1:length(Pc)
    i
    demanda=Pc(3,i);
    w0=Pc(2,i);
    teta0=Pc(1,i);
    sim('LRM1',[0 50e-3]);
    dteta=tetaf-teta0;
    dw=wf-w0;
Tc=[Tc [dteta ;dw]];
end
%Exportacion de los datos
filename = 'InMRLAntena.csv'; csvwrite(filename,Pc)
filename = 'OutMRLAntena.csv'; csvwrite(filename,Tc)
```

Figura 131. Adquisición de patrones de entrenamiento – MRL Antena

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy. Se debe tomar en cuenta que este modelo consta de dos redes neuronales acopladas, una es el controlador que se desea adquirir y la otra es el sistema ya identificado previamente y en la cual los pesos no se deben modificar.

Para el entrenamiento de este modelo, se necesita extraer las variaciones de estado para concatenar con la salida del controlador e ingresar a la red neuronal del sistema identificado previamente.

```
#Se importa de los archivos exportados desde MATLAB
Pm = np.matrix(pd.read_csv("InMRLAntena.csv", header=None).values)
Tm = np.matrix(pd.read_csv("OutMRLAntena.csv", header=None).values)

#Se hace un reshape
Pm= Pm.T.reshape(-1,3)
Tm= Tm.T.reshape(-1,2)

#Se separa de todos los patrones de entrenamiento, para poder hacer
#un test para ver el rendimiento de la red
X_train, X_test, y_train, y_test = train_test_split(Pm, Tm, test_size=0.20, random_state=40)
#Se extrae las variaciones de estado para la red de identificacion
Tm2=np.concatenate((X_train[:,[0,1]]))
Xt2=np.concatenate((X_test[:,[0,1]]))
```

Figura 132. Patrones de entrenamiento – MRL Antena

Importación del modelo del sistema identificado

Se realiza la importación del modelo del sistema previamente identificado y se extrae los pesos y los bias del mismo para poder pasarlo a nuestra nueva red a entrenar.

```
#CARGAR MODELO MODELNETWORK
modelide = load_model('IdentAntena.h5')
#TOMAR LOS PESOS de MODELNETWORK
PyB=np.asarray([modelide.get_weights()]).T.reshape(-1,1)
Pc1=PyB[0,:]
Bc1=PyB[1,:]
Pc2=PyB[2,:]
Bc2=PyB[3,:]
Pc3=PyB[4,:]
Bc3=PyB[5,:]

Pc1=Pc1[0]
Bc1=Bc1[0]
Pc2=Pc2[0]
Bc2=Bc2[0]
Pc3=Pc3[0]
Bc3=Bc3[0]
```

Figura 133. Importación Modelo Identificado – MRL Antena

Creación de modelo

Para el entrenamiento de este problema se va a utilizar un modelo funcional, ya que se necesita especificar la conexión entre las capas y realizar la concatenación de la salida del controlador con las variaciones de las variables de estado.

Al momento de agregar las capas del modelo ya identificado, se debe tomar en cuenta que, las dimensiones de las mismas deben coincidir, es decir, deben ser iguales a las dimensiones de las capas que se añadieron cuando se realizó la identificación del sistema. Además, se debe configurar para que los pesos y bias de estas capas no se actualicen mientras se realiza el entrenamiento. Para esto se añade el parámetro “trainable=0”

Además de no permitir que los pesos se actualicen, necesitan estos ser inicializados con los pesos y bias del sistema ya identificado. Para esto se añade el parámetro “weights=[Pesos,Bias]”, según corresponda.

```
#Controlador MRL
visible1 = Input(shape=(3,))
dense11 = Dense(8,input_shape=X_train.shape)(visible1)
Leaky11 = LeakyReLU(alpha=0.085)(dense11)
dense12 = Dense(1)(Leaky11)
visible2 = Input(shape=(2,))
# Unir entrada de red neuronal y estados
merge = concatenate([visible2, dense12])
# ModelNetwork
hidden1 = Dense(8, activation='relu',weights=[Pc1,Bc1],trainable=0,input_shape
=merge.shape)(merge)
hidden2 =(Dense(8,activation='relu',weights=[Pc2,Bc2],trainable=0))(hidden1)
output = Dense(2,weights=[Pc3,Bc3],trainable=0)(hidden2)
model = Model(inputs=[visible1, visible2], outputs=output)
```

Figura 134. Creación del modelo en Python – MRL Antena

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 3)	0	
dense_3 (Dense)	(None, 8)	32	input_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 8)	0	dense_3[0][0]
input_3 (InputLayer)	(None, 2)	0	
dense_4 (Dense)	(None, 1)	9	leaky_re_lu_2[0][0]
concatenate_1 (Concatenate)	(None, 3)	0	input_3[0][0] dense_4[0][0]
dense_5 (Dense)	(None, 8)	32	concatenate_1[0][0]
dense_6 (Dense)	(None, 8)	72	dense_5[0][0]
dense_7 (Dense)	(None, 2)	18	dense_6[0][0]

Total params: 163
Trainable params: 41
Non-trainable params: 122

None

Figura 135. Resumen red neuronal – MRL Antena

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 136. Compilación de red neuronal identificación – MRL Antena

Entrenamiento del modelo

Para el entrenamiento de la red neuronal se ha considerado 600 épocas. Para este caso se debe considerar el formato de los patrones de entrenamiento de entrada.

```
history = model.fit([X_train,Tm2], y_train, epochs=600)
```

Figura 137. Entrenamiento de red neuronal identificación – MRL Antena

Extracción de los pesos y bias del controlador

Para esto se usa el siguiente código:

```
#C#TOMAR LOS PESOS de NUEVO CONTROLADOR
PyBm=np.asarray([model.get_weights()]).T.reshape(-1,1)
Pc1m=PyBm[0,:]
Bc1m=PyBm[1,:]
Pc2m=PyBm[2,:]
Bc2m=PyBm[3,:]

Pc1m=Pc1m[0]
Bc1m=Bc1m[0]
Pc2m=Pc2m[0]
Bc2m=Bc2m[0]
```

Figura 138. Extracción de Pesos y Bias de controlador – MRL Antena

Creación de modelo para controlador

Se necesita crear un modelo en el cual podamos pasar los pesos y bias extraidos previamente para posteriormente exportar el modelo. Se debe tomar en cuenta que las dimensiones de las capas deben ser las mismas para poder pasar los pesos.

En este caso se usará un modelo secuencial:

```
modelcontrol = Sequential()
modelcontrol.add(Dense(8,input_dim=3,weights=[Pc1m,Bc1m],tr
ainable=0))
modelcontrol.add(LeakyReLU(alpha=0.085))
modelcontrol.add(Dense(1,weights=[Pc2m,Bc2m],trainable=0))
```

Figura 139. Creación de modelo para controlador – MRL Antena

Compilación de modelo para controlador

Ya que este modelo se usará como recipiente en el cual solo se copiará los pesos y bias. Solo se realizará la compilación para definir el modelo, omitiendo el entrenamiento.

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Exportación del modelo

```
#GUARDAR MODELO
from keras.models import load_model
modelcontrol.save('MRLAntenaControl.h5')
```

Figura 140. Exportación de red neuronal – MRL Antena

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “MRLControlAntena.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El archivo que contiene el modelo “MRLAntenaControl.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = MRLControlAntena(teta0,w0,u)
 modelfile = 'MRLAntenaControl.h5';
 net = importKerasNetwork(modelfile);
 yu=predict(net,[teta0 w0 u]);
 yu=double(yu);
y = yu;
```

Figura 141. Función en MATLAB – MRL Antena

Resultados

Se presenta el modelo exportado en imagen formato png.

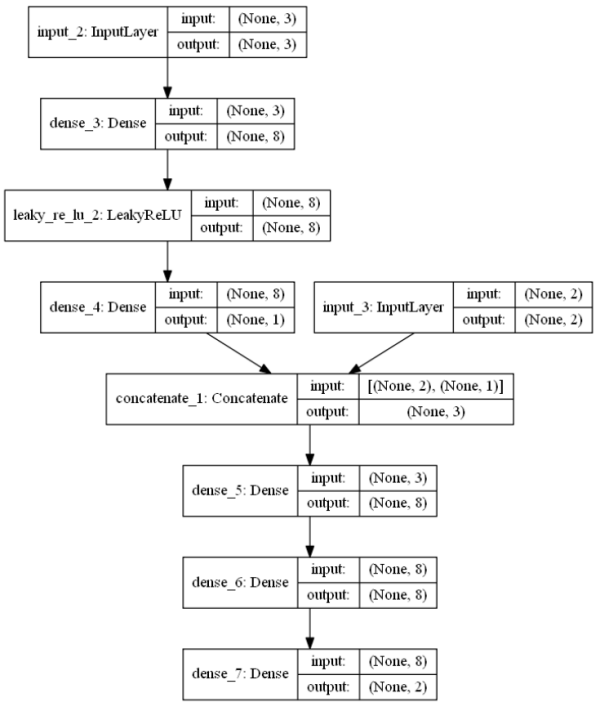


Figura 142. Modelo RN – MRL Antena

Se realiza la simulación en paralelo del sistema en diagrama de bloques y de la red neuronal.

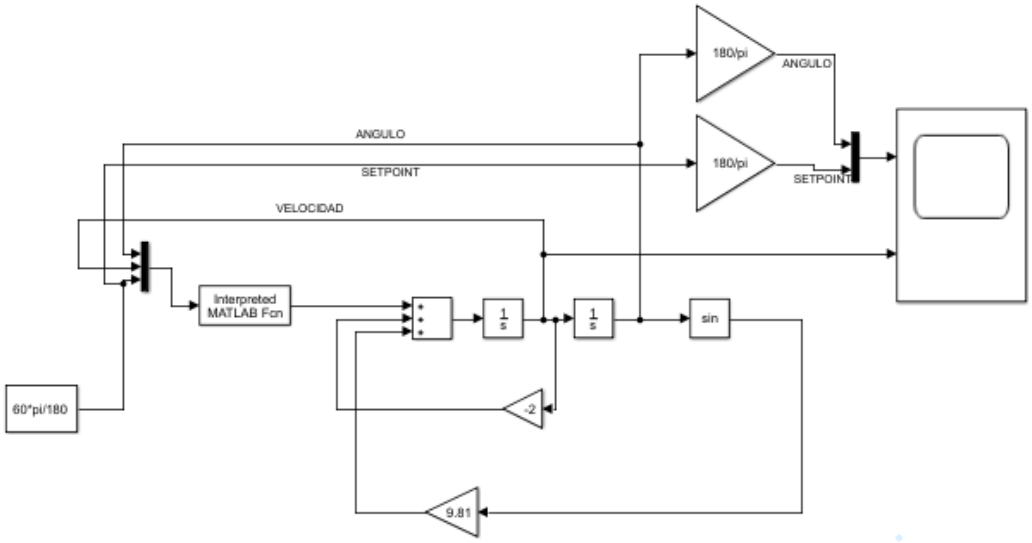


Figura 143. Sistema controlado – MRL Antena

Se propone un SetPoint de 60 grados, con condiciones iniciales de $\theta = -10$ grados y $\omega = 0$ (grados/seg).

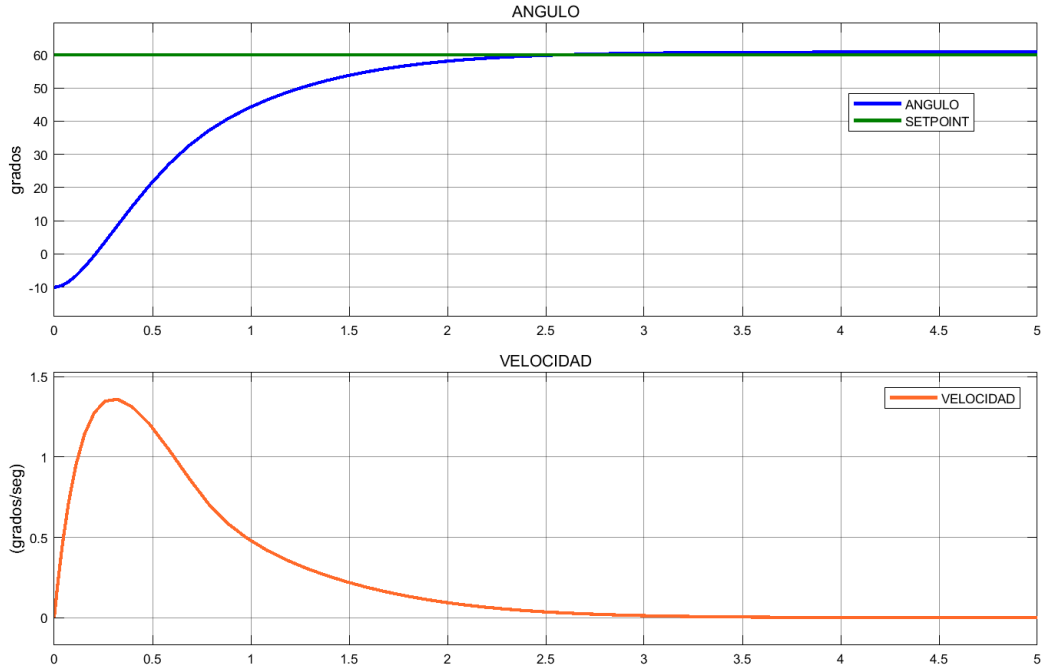


Figura 144. Curvas de variables de estado sistema controlado – MRL Antena

3.4.2. Control con modelo de referencia para el sistema Tanques Acoplados

Para la realización del controlador con modelo de referencia lineal, se es necesario tener las ecuaciones que respondan a este modelo, pero cuando no se dispone de las mismas, se recurre a realizar el control del sistema por medio de PID, y a este sistema controlado se lo hace actuar como el modelo de referencia lineal, como se muestra a continuación:

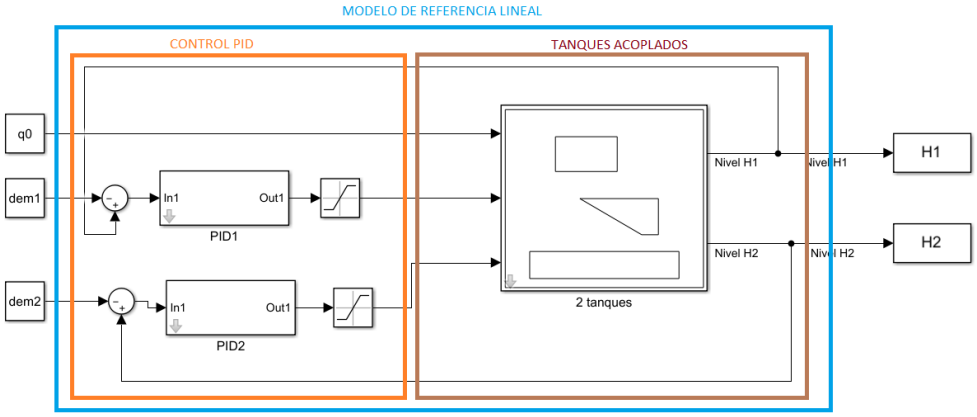


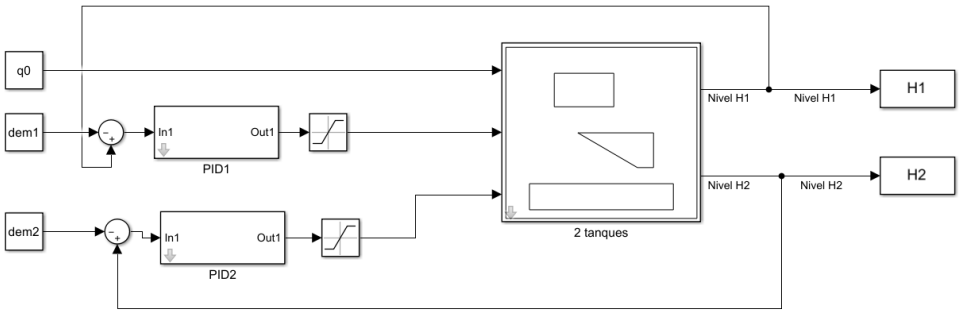
Figura 145. Sistema controlado por PID – MRL Tanques

Ya con el sistema controlado se procede a la generación de los patrones de entrenamiento.

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para el controlador con modelo de referencia lineal

Para obtener los ejemplos de entrenamiento, agregamos bloques de entrada y de salida para la obtención de las variaciones de las variables de estado. Se debe tomar en cuenta que el sistema de tanques acoplados es un sistema lento, por lo tanto, es recomendable realizar las simulaciones con más de 1 segundo de duración para cada combinación posible.




```
iniV=[1:5:26];%Posibles alturas tanque 1
H2iniV=[1:5:26]%Posibles alturas tanque 2
dem1V=[3:5:23] %demanda h1
dem2V=[3:5:23] %demanda h2
q=[10:9:37] %- caudal (m3/s)
%Combinacion de valores de vectores : combvec
Pm=combvec(H1iniV,dem1V,H2iniV,dem2V,q);
Tm=[]
for i=1:length(Pm)
  H1ini=Pm(1,i);
  dem1=Pm(2,i);
  H2ini=Pm(3,i);
  dem2=Pm(4,i);
  q0=Pm(5,i);
  sim('IDE_two_tank_pid',[0 1]);
  dh1=H1-H1ini;
  dh2=H2-H2ini;
  Tm=[Tm [dh1 ;dh2]];
end
filename = 'InMRLTanque.csv'; csvwrite(filename,Pm')
filename = 'OutMRLTanque.csv'; csvwrite(filename,Tm')
```

Figura 146. Adquisición de patrones de entrenamiento MATLAB – MRL Tanques

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy. Se debe tomar en cuenta que este modelo consta de dos redes neuronales acopladas, una es el controlador que se desea adquirir y la otra es el sistema ya identificado previamente y en la cual los pesos no se deben modificar.

Para el entrenamiento de este modelo, se necesita extraer las variaciones de estado para concatenar con la salida del controlador e ingresar a la red neuronal del sistema identificado previamente.

```
#Se importa de los archivos exportados desde MATLAB
Pm = np.matrix(pd.read_csv("InMRLTanque.csv", header=None).values)
Tm = np.matrix(pd.read_csv("OutMRLTanque.csv", header=None).values)
#Se separa de todos los patrones de entrenamiento, para poder hacer
#un test para ver el rendimiento de la red
X_train, X_test, y_train, y_test = train_test_split(Pm, Tm, test_size=0.20, random_state=40)
#Se extrae las variaciones de estado para la red de identificacion
H1=np.concatenate((X_train[:,[0]]))
H2=np.concatenate((X_train[:,[2]]))
O=np.concatenate((X_train[:,[4]]))
```

Figura 147. Patrones de entrenamiento – MRL Tanques

Normalización de las entradas

El objetivo de la normalización es hacer que las entradas tengan un valor entre 0 y 1 (lo cual suele ayudar en el entrenamiento de las redes neuronales).

Para la normalización de las entradas se ha hecho uso del siguiente código:

```
X_train=np.concatenate((X_train[:,[0]]*1/26,X_train[:,[1]]*1/23,X_train[:,[2]]*1
/26,X_train[:,[3]]*1/23,X_train[:,[4]]*1/37),axis=1)
```

Figura 148. Entradas normalizadas – MRL Tanques

Importación del modelo del sistema identificado

Se realiza la importación del modelo del sistema previamente identificado y se extrae los pesos y los bias del mismo para poder pasarlo a nuestra nueva red a entrenar.

```
#CARGAR MODELO MODELNETWORK
modelide = load_model('IdentAntena.h5')
#TOMAR LOS PESOS de MODELNETWORK
PyB=np.asarray([modelide.get_weights()]).T.reshape(-1,1)
Pc1=PyB[0,:]
Bc1=PyB[1,:]
Pc2=PyB[2,:]
Bc2=PyB[3,:]

Pc1=Pc1[0]
Bc1=Bc1[0]
Pc2=Pc2[0]
Bc2=Bc2[0]
```

Figura 149. Importación Modelo Identificado – MRL Tanques

Creación de modelo

En este problema la salida del controlador va a ser de dos dimensiones (apertura de la válvula 1 y apertura de la válvula 2). Ya que el modelo de los tanques necesita un orden en específico de entradas, se han creado dos capas (ap1, ap2), las cuales se conectarán a la capa anterior. Esto nos servirá para poder concatenar en el orden especifico que el modelo ya identificado necesita.

```
#Controlador MRL
visible1 = Input(shape=(5,))
dense11 = Dense(16,input_shape=X_train.shape,activation='relu')(visible1)
ap1=Dense(1)(dense11)
ap2=Dense(1)(dense11)

# Se instancia las entradas para MODELNETWORK
EH1=Input(shape=(1,))
EH2=Input(shape=(1,))
Eq=Input(shape=(1,))

# Unir entrada de red neuronal y estados
merge = concatenate([EH1,ap1,EH2,ap2,Eq])
# ModelNetwork
hidden1 = Dense(8, activation='tanh',weights=[Pc1,Bc1],trainable=0,input_shape=mer
ge.shape)(merge)
output = Dense(2,weights=[Pc3,Bc3],trainable=0)(hidden1)
model = Model(inputs=[visible1,EH1,EH2,Eq], outputs=output)
```

Figura 150. Creación del modelo en Python – MRL Tanques

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 5)	0	
dense_1 (Dense)	(None, 16)	96	input_1[0][0]
input_2 (InputLayer)	(None, 1)	0	
dense_2 (Dense)	(None, 1)	17	dense_1[0][0]
input_3 (InputLayer)	(None, 1)	0	
dense_3 (Dense)	(None, 1)	17	dense_1[0][0]
input_4 (InputLayer)	(None, 1)	0	
concatenate_1 (Concatenate)	(None, 5)	0	input_2[0][0] dense_2[0][0] input_3[0][0] dense_3[0][0] input_4[0][0]
dense_4 (Dense)	(None, 8)	48	concatenate_1[0][0]
dense_5 (Dense)	(None, 2)	18	dense_4[0][0]

Total params: 196
Trainable params: 130
Non-trainable params: 66

Figura 151. Resumen red neuronal – MRL Tanques

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer=RMSprop, metrics=['accuracy'])
```

Figura 152. Compilación de red neuronal identificación – MRL Tanques

Entrenamiento del modelo

Para el entrenamiento de la red neuronal se ha considerado 600 épocas. Para este caso se debe considerar el formato de los patrones de entrenamiento de entrada.

```
1.  history = model.fit([X_train,H1,H2,Q] , y_train, epochs=600)
```

Figura 153. Entrenamiento de red neuronal identificación – MRL Tanques

Extracción de los pesos y bias del controlador

Para esto se usa el siguiente código:

```
#C#TOMAR LOS PESOS de NUEVO CONTROLADOR
PyBm=np.asarray([model.get_weights()]).T.reshape(-1,1)
Pc1m=PyBm[0,:]
Bc1m=PyBm[1,:]
Pc2m=PyBm[2,:]
Bc2m=PyBm[3,:]
Pc3m=PyBm[4,:]
Bc3m=PyBm[5,:]

Pc1m=Pc1m[0]
Bc1m=Bc1m[0]
Pc2m=Pc2m[0]
Bc2m=Bc2m[0]
Pc3m=Pc3m[0]
Bc3m=Bc3m[0]
```

Figura 154. Extracción de Pesos y Bias de controlador – MRL Tanques

Como se ha separado la salida del controlador en dos capas de una sola neurona, se debe concatenar los pesos y los bias de las mismas antes de pasar al nuevo modelo "contenedor" para poder exportarlo.

```
#C#TOMAR LOS PESOS
Pct=np.concatenate((Pc2m,Pc3m)).T.reshape(-1,16)
Pct=Pct.T.reshape(-1,2)
Bct=np.concatenate((Bc2m,Bc3m)).T.reshape(-1,)
```

Figura 155. Concatenación de pesos de la última capa del controlador – MRL Tanques

Creación de modelo para controlador

Se necesita crear un modelo en el cual podamos pasar los pesos y bias extraidos previamente para posteriormente exportar el modelo. Se debe tomar en cuenta que las dimensiones de las capas deben ser las mismas para poder pasar los pesos.

En este caso se usará un modelo secuencial:

```
modelcontrol = Sequential()
modelcontrol.add(Dense(16,input_dim=3,weights=[Pc1m,Bc1m],trainable=0,
activation='relu'))
modelcontrol.add(Dense(2,weights=[Pct,Bct],trainable=0))
```

Figura 156. Creación de modelo para controlador – MRL Tanques

Compilación de modelo para controlador

Ya que este modelo se usará como recipiente en el cual solo se copiará los pesos y bias. Solo se realizará la compilación para definir el modelo, omitiendo el entrenamiento.

```
2.  model.compile(loss='mean_squared_error', optimizer=RMSprop, metrics=['accuracy'])
```

Exportación del modelo

```
7.  #GUARDAR MODELO
8.  from keras.models import load_model
9.  modelcontrol.save('MRLTanqueControl.h5')
```

Figura 157. Exportación de red neuronal – MRL Tanques

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “MRLTanque.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El archivo que contiene el modelo “MRLTanqueControl.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = MRLTanque(h1,v1,h2,v2,q)
 modelfile = 'MRLTanqueControl.h5';
 net = importKerasNetwork(modelfile);
 yu=predict(net,[h1 v1 h2 v2 q]);
 yu=double(yu);
y = yu;
```

Figura 158. Función en MATLAB neuronal – MRL Tanques

Resultados

Se presenta el modelo exportado en imagen formato png.

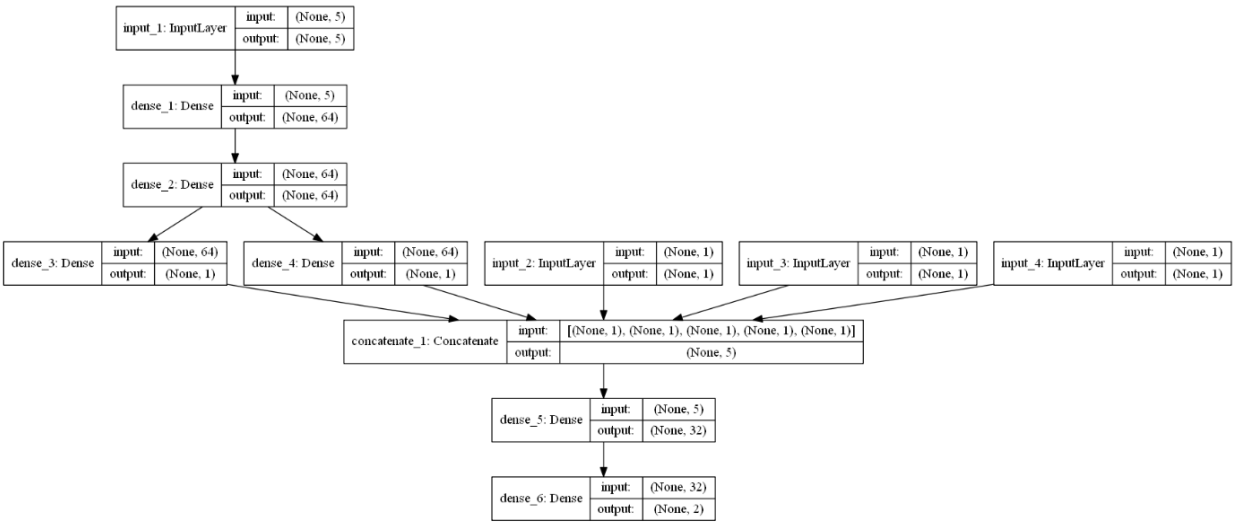


Figura 159. Modelo RN – MRL Tanques

Se realiza la simulación en paralelo del sistema en diagrama de bloques y de la red neuronal.

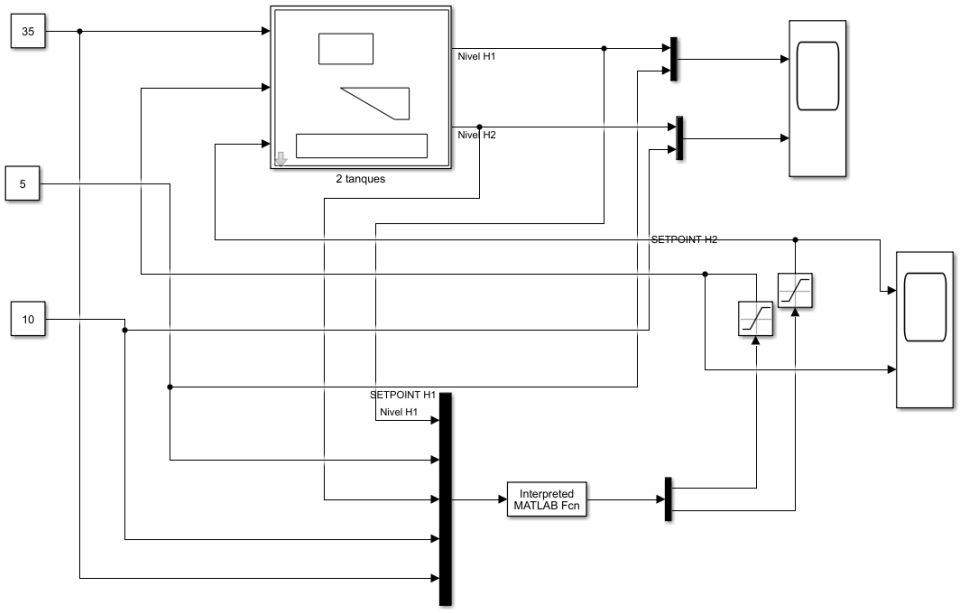


Figura 160. Sistema controlado – MRL Tanques

Se propone un SetPoint para la altura del tanque 1 de 20 cm y de 25 cm para tanque 2, con condiciones iniciales de $h1 = 3$ cm y $h2 = 5$ cm y de $q = 20\ \left(\frac{m^3}{s}\right)$

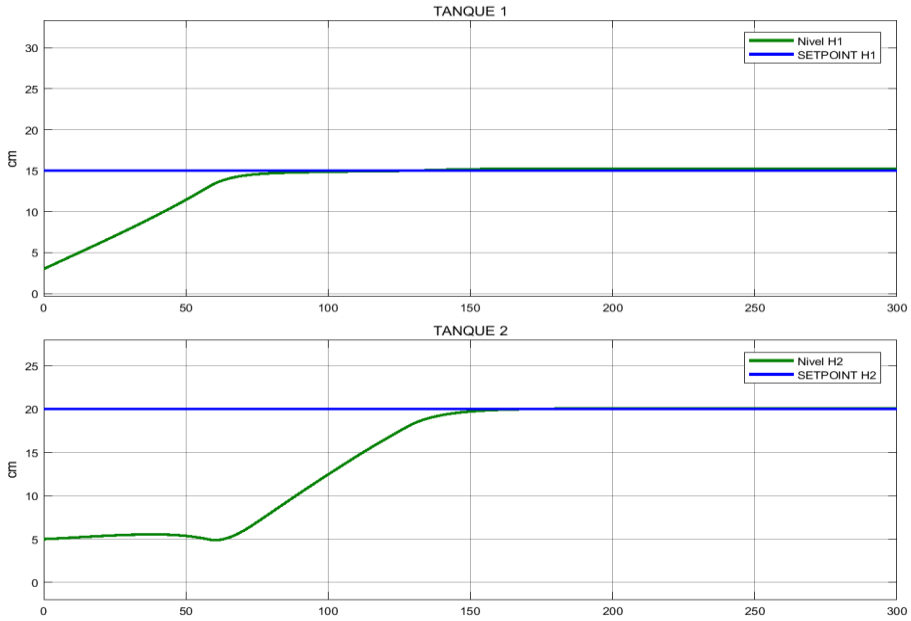


Figura 161. Curvas de variables de estado sistema controlado – MRL Tanques

3.4.3. Control con modelo de referencia para el sistema Viga y Bola

Para la realización del controlador con modelo de referencia lineal, al igual que en el sistema de Tanques Acoplados se realiza el control del sistema por medio de PID, y a este sistema controlado se lo hace actuar como el modelo de referencia lineal, como se muestra a continuación:

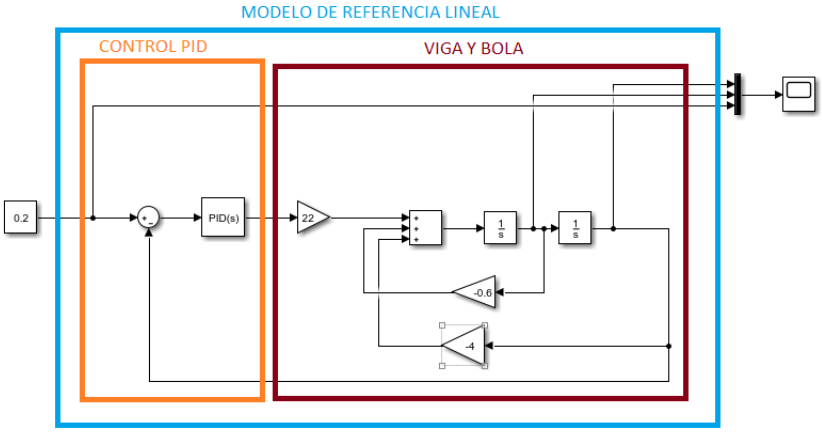


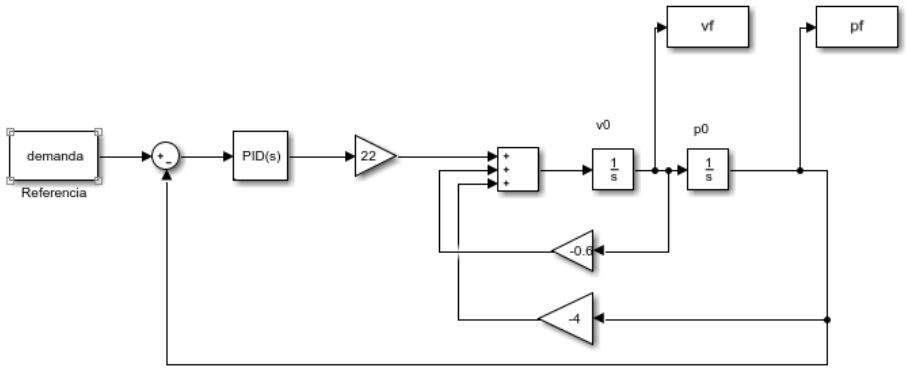
Figura 162. Sistema controlado por PID – MRL Viga y Bola

Ya con el sistema controlado se procede a la generación de los patrones de entrenamiento.

Elaboración de la red neuronal

Adquisición de patrones de entrenamiento para el controlador con modelo de referencia lineal

Para obtener los ejemplos de entrenamiento, agregamos bloques de entrada y de salida para la obtención de las variaciones de las variables de estado.




```
vel=[-0.7:0.2:0.7];
pos=0:0.2:1;
pos2=0:0.1:1;
demanda=0:0.1:0.99;
Pc = [combvec(pos,vel,demanda) [pos2; zeros(2,length(pos2))]];
Tc=[]
for i=1:length(Pc)
    demanda=Pc(3,i);
    v0=Pc(2,i);
    p0=Pc(1,i);
    sim('IdeRMLViga',[0 50e-3]);
    dp=pf-p0;
    dv=vf-v0;
Tc=[Tc [dp ;dv]];
end

filename = 'InMRLViga.csv'; csvwrite(filename,Pc)
filename = 'OutMRLViga.csv'; csvwrite(filename,Tc)
```

Figura 163. Adquisición de patrones de entrenamiento – MRL Viga y Bola

Patrones de entrenamiento

Se importan los patrones de entrenamiento desde los archivos exportados previamente desde MATLAB con la ayuda de la librería Numpy. Se debe tomar en cuenta que este modelo consta de dos redes neuronales acopladas, una es el controlador que se desea adquirir y la otra es el sistema ya identificado previamente y en la cual los pesos no se deben modificar.

Para el entrenamiento de este modelo, se necesita extraer las variaciones de estado para concatenar con la salida del controlador e ingresar a la red neuronal del sistema identificado previamente.

```
#Se importa de los archivos exportados desde MATLAB
Pm = np.matrix(pd.read_csv("InMRLViga.csv", header=None).values)
Tm = np.matrix(pd.read_csv("OutMRLViga.csv", header=None).values)
#Transpuesta
Pm= Pm.T.reshape(-1,3)
Tm= Tm.T.reshape(-1,2)
#Se separa de todos los patrones de entrenamiento, para poder hacer
#un test para ver el rendimiento de la red
X_train, X_test, y_train, y_test = train_test_split(Pm, Tm, test_size=0.20, random_state=40)
#Se extrae las variaciones de estado para la red de identificacion
Tm2=np.concatenate((X_train[:,[0,1]]))
Xt2=np.concatenate((X_test[:,[0,1]]))
```

Figura 164. Patrones de entrenamiento – MRL Viga y Bola

Importación del modelo del sistema identificado

Se realiza la importación del modelo del sistema previamente identificado y se extrae los pesos y los bias del mismo para poder pasarlo a nuestra nueva red a entrenar.

```
#CARGAR MODELO MODELNETWORK
modelide = load_model('IdentAntena.h5')
#TOMAR LOS PESOS de MODELNETWORK
PyB=np.asarray([modelide.get_weights()]).T.reshape(-1,1)
Pc1=PyB[0,:]
Bc1=PyB[1,:]
Pc2=PyB[2,:]
Bc2=PyB[3,:]
Pc3=PyB[4,:]
Bc3=PyB[5,:]

Pc1=Pc1[0]
Bc1=Bc1[0]
Pc2=Pc2[0]
Bc2=Bc2[0]
Pc3=Pc3[0]
Bc3=Bc3[0]
```

Figura 165. Importación Modelo Identificado – MRL Viga y Bola

Creación de modelo

El programa para la creación del modelo es el siguiente:

```
#Controlador MRL
visible1 = Input(shape=(3,))
dense11 = Dense(8,input_shape=X_train.shape)(visible1)
Leaky11 = LeakyReLU(alpha=0.35)(dense11)
dense12 = Dense(4)(Leaky11)
dense13 = Dense(1)(dense12)
visible2 = Input(shape=(2,))
# Unir entrada de red neuronal y estados
merge = concatenate([visible2, dense13])
# ModelNetwork
hidden1 = Dense(8, activation='relu',weights=[Pc1,Bc1],trainable=0,input_shape
=merge.shape)(merge)
hidden2 =(Dense(8,activation='relu',weights=[Pc2,Bc2],trainable=0))(hidden1)
output = Dense(2,weights=[Pc3,Bc3],trainable=0)(hidden2)
```

Figura 166. Creación del modelo en Python – MRL Viga y Bola

Si se ejecuta el comando `model.summary()` se puede visualizar un resumen de lo que es nuestro modelo propuesto, donde se puede apreciar el tipo de capa, la dimensión de la salida y los parámetros entrenables y no entrenables de cada capa.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 3)	0	
dense_1 (Dense)	(None, 8)	32	input_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 8)	0	dense_1[0][0]
dense_2 (Dense)	(None, 4)	36	leaky_re_lu_1[0][0]
input_2 (InputLayer)	(None, 2)	0	
dense_3 (Dense)	(None, 1)	5	dense_2[0][0]
concatenate_1 (Concatenate)	(None, 3)	0	input_2[0][0] dense_3[0][0]
dense_4 (Dense)	(None, 8)	32	concatenate_1[0][0]
dense_5 (Dense)	(None, 8)	72	dense_4[0][0]
dense_6 (Dense)	(None, 2)	18	dense_5[0][0]

Total params: 195
Trainable params: 73
Non-trainable params: 122

Figura 167. Resumen red neuronal – MRL Viga y Bola

Compilación de modelo

Para la compilación se utiliza el error cuadrático medio como función de pérdida, Adam como optimizador y accuracy para la métrica de evaluación de la red neuronal.

```
1.  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 168. Compilación de red neuronal – MRL Viga y Bola

Entrenamiento del modelo

Para el entrenamiento de la red neuronal se ha considerado 900 épocas. Para este caso se debe considerar el formato de los patrones de entrenamiento de entrada.

```
1.  history = model.fit([X_train,Tm2], y_train, epochs=900)
```

Figura 169. Entrenamiento de red neuronal – MRL Viga y Bola

Extracción de los pesos y bias del controlador

Para esto se usa el siguiente código:

```
#C#TOMAR LOS PESOS de NUEVO CONTROLADOR
PyBm=np.asarray([model.get_weights()]).T.reshape(-1,1)
Pc1m=PyBm[0,:]
Bc1m=PyBm[1,:]
Pc2m=PyBm[2,:]
Bc2m=PyBm[3,:]

Pc1m=Pc1m[0]
Bc1m=Bc1m[0]
Pc2m=Pc2m[0]
Bc2m=Bc2m[0]
```

Figura 170. Extracción de Pesos y Bias de controlador – MRL Viga y Bola

Creación de modelo para controlador

Se necesita crear un modelo en el cual podamos pasar los pesos y bias extraidos previamente para posteriormente exportar el modelo. Se debe tomar en cuenta que las dimensiones de las capas deben ser las mismas para poder pasar los pesos.

En este caso se usará un modelo secuencial:

```
modelcontrol = Sequential()
modelcontrol.add(Dense(8,input_dim=3,weights=[Pc1m,Bc1m],trainable=0))
modelcontrol.add(LeakyReLU(alpha=0.35))
modelcontrol.add(Dense(4,weights=[Pc2m,Bc2m],trainable=0))
modelcontrol.add(Dense(1,weights=[Pc3m,Bc3m],trainable=0))
```

Figura 171. Creación de modelo para controlador – MRL Viga y Bola

Compilación de modelo del controlador

Ya que este modelo se usará como “recipiente” en el cual solo se copiará los pesos y bias. Solo se realizará la compilación para definir el modelo, omitiendo el entrenamiento.

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figura 172. Compilación modelo del controlador – MRL Viga y Bola

Exportación del modelo

```
1.  #GUARDAR MODELO
2.  from keras.models import load_model
3.  modelcontrol.save('MRLVigaControl.h5')
```

Figura 173. Exportación de red neuronal – MRL Viga y Bola

Importación del modelo en MATLAB

Con el objetivo de usar este modelo en las simulaciones de Simulink, se ha creado una función “MRLControlViga.m” de MATLAB, donde se llama al modelo y cada vez que se lo hace, este regresa un valor flotante que es resultado de la predicción con valores de entrada específicos. El archivo que contiene el modelo “MRLVigaControl.h5”, debe estar en el mismo directorio de la función de MATLAB.

```
function y = MRLControlViga(p,v,demanda)
modelfile = 'MRLVigaControl.h5';
net = importKerasNetwork(modelfile);
yu=predict(net,[p v demanda]);
yu=double(yu);
y = yu;
```

Figura 174. Función en MATLAB – MRL Viga y Bola

Resultados

Se presenta el modelo exportado en imagen en formato png.

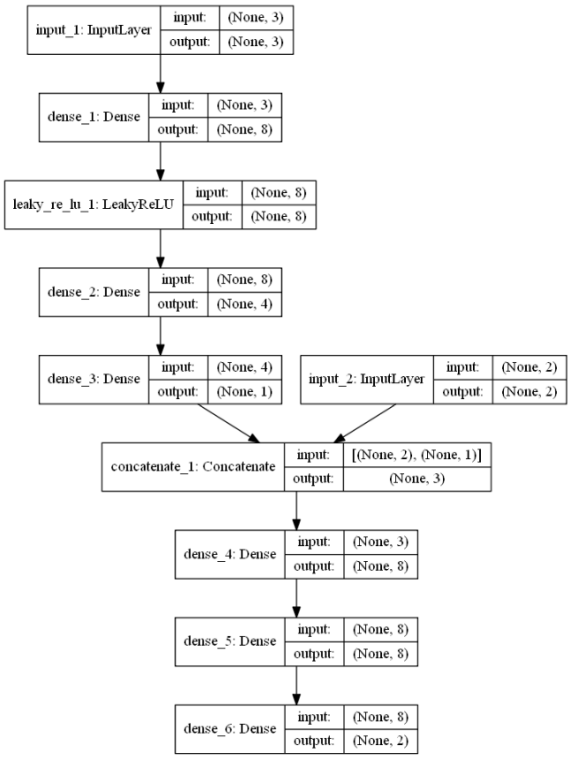


Figura 175. Modelo RN – MRL Viga y Bola

Se realiza la simulación del controlador MRL, como se muestra:

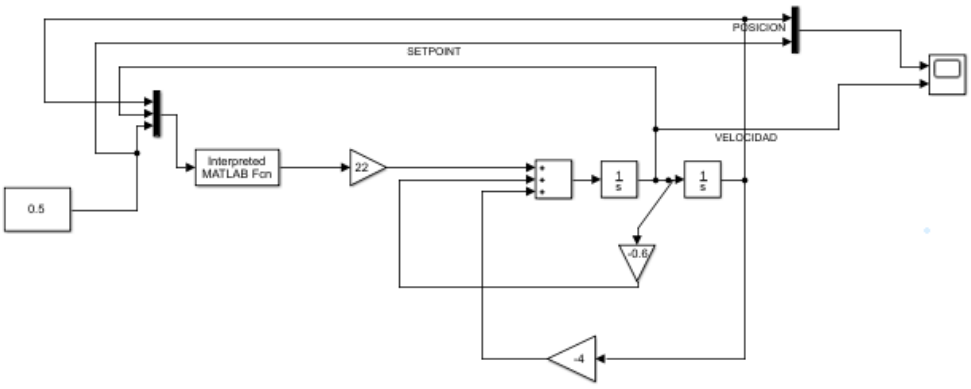


Figura 176. Sistema controlado – MRL Viga y Bola

Se propone un setpoint de 50 cm (0.5m), con condiciones iniciales de $p = 0$ cm y $v = 0$ (cm/seg).

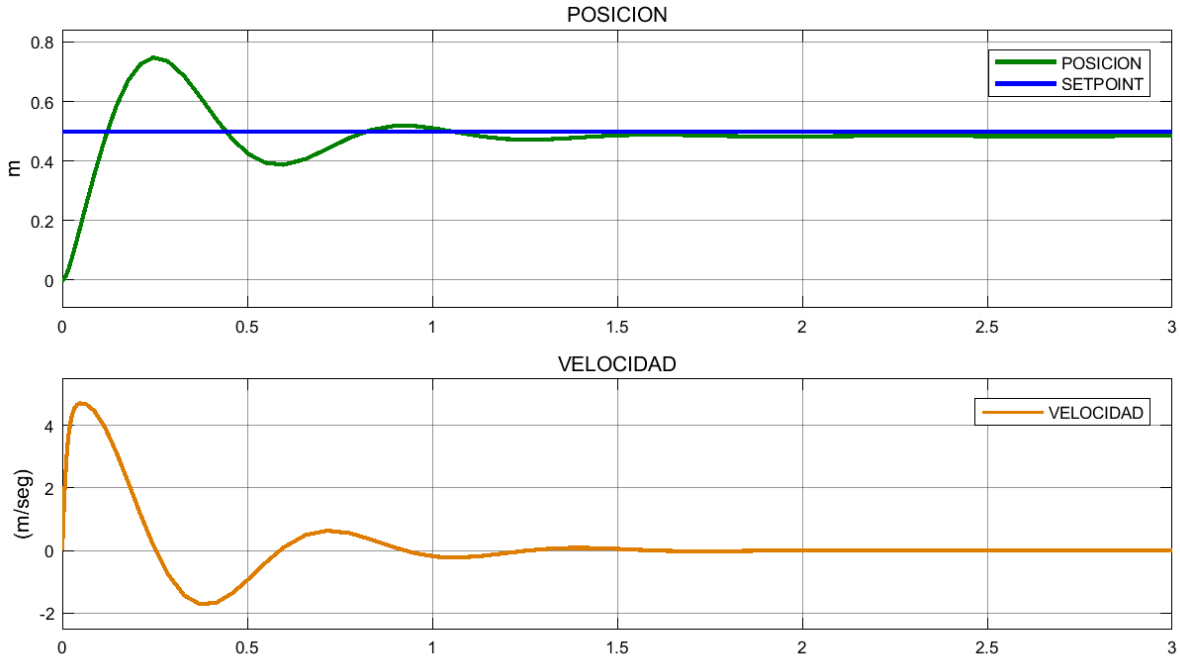


Figura 177. Curvas de variables de estado sistema controlado – MRL Viga y Bola

CAPÍTULO IV

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

Se desarrollaron aplicaciones didácticas haciendo uso de la librería TensorFlow para resolver problemas de control realimentado de procesos mediante redes neuronales de aprendizaje profundo. Se desarrolló todas las aplicaciones con la librería TensorFlow, utilizando sus API (Application Programming Interface), su estructura, sus modelos de entrenamiento y principales funciones. Para experimentar con las soluciones se aplicó la API Keras, de TensorFlow, que nos permite crear modelos de redes neuronales secuenciales y funcionales y exportarlas al entorno de Matlab.

Se desarrolló aplicaciones básicas de operaciones lógicas y de identificación de funciones mediante redes neuronales con aprendizaje profundo. Para la identificación y el desarrollo de los controladores neuronales se utilizó la ecuación dinámica que describe a cada sistema. La obtención de los patrones de entrenamiento requiere definir los rangos que las variables de estado y la señal de control pueden tomar para caracterizar el sistema en las posibles condiciones de funcionamiento.

Se usó dos sistemas dinámicos controlados por un controlador PID (para el sistema de Tanques Acoplados y el sistema Viga y Bola) como Modelos de Referencia Lineal, para desarrollar el Control Neuronal por Modelo de Referencia. Se logró de manera efectiva la identificación, el control por red neuronal inversa y el control por modelo de referencia lineal de cada uno de los sistemas retroalimentados propuestos.

Se determinó que Adam es efectivo como optimizador y por ello es el más recomendado para el entrenamiento de modelos de aprendizaje profundo, ya que sus parámetros de configuración predeterminados funcionan bien en la mayoría de los problemas.

4.2. Recomendaciones

Se recomienda empezar por un modelo con pocas neuronas y función de activación ReLU. Si los patrones de entrenamiento son en gran cantidad, se recomienda usar el optimizador Adam. Se recomienda analizar el tipo de sistema (rápido o lento) del que se desea adquirir los patrones de entrenamiento, con el objetivo de escoger un tiempo adecuado de discretización. Si el sistema es lento (p. ej., Planta de Tanques acoplados), un tiempo adecuado es mayor a 1 segundo y si el sistema es rápido (p. ej., Péndulo Invertido), un tiempo adecuado sería menor a 1 segundo.

Mientras se realiza el entrenamiento se recomienda observar si hay variación en la métrica de evaluación para poder establecer un número prudente de épocas y así evitar el sobreajuste en la red neuronal.

Antes de realizar el control neuronal inverso, se recomienda hacer la simulación del sistema identificado con el sistema real y observar si se comportan de forma similar.

Para la adquisición de patrones de entrenamiento para el control por Modelo de referencia Lineal, se recomienda usar un control PID y sintonizarlo con la ayuda del Toolbox de MATLAB.

CAPÍTULO V

5. BIBLIOGRAFIA

Calvo, D. (n.d.). Función de activación - Redes neuronales. Retrieved December 8, 2019, from http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

Gurney, K. (1997). An Introduction to Neural Networks. In *An Introduction to Neural Networks*. https://doi.org/10.7551/mitpress/3905.001.0001

Kriesel, D. (2009). *Neural Networks*. *5*(1), 3–8. Retrieved from http://www.dkriesel.com/en/science/neural_networkshttp://www.dkriesel.com/en/tech/snipe

Kukreja, H., N, B., Siddesh, C. S., & Kuldeep, S. (2016). (PDF) An Introduction to Artificial Neural Network. Retrieved December 6, 2019, from https://www.researchgate.net/publication/319903816_AN_INTRODUCTION_TO_ARTIFICIAL_NEURAL_NETWORK

Marín Diazaraque, J. M. (2007). Introducción a las redes neuronales aplicadas. *Manual Data Mining*, *3*, 1–31. Retrieved from http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/tema3dm.pdf

Meiss, J. (2007). Dynamical systems. In *Scholarpedia* (Vol. 2). https://doi.org/10.4249/scholarpedia.1629

Nelles, O., & Nelles, O. (2001). Introduction. In *Nonlinear System Identification* (pp. 1–19). https://doi.org/10.1007/978-3-662-04323-3_1

Tim Jones, M. (2017). Models for machine learning – IBM Developer. Retrieved December 6, 2019, from https://developer.ibm.com/articles/cc-models-machine-learning/

Verdult, V. (2002). *Nonlinear System Identification: A State-Space Approach*. Retrieved from http://www.tup.utwente.nl/catalogue/book/index.jsp?isbn=9036517176