TRABAJO FIN DE GRADO

INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

Controlador Neuronal

Esquemas de aprendizaje de modelos internos de brazo robótico con múltiples articulaciones

Autor
Julio Jesús Vizcaíno Molina

Directores
Jesús Garrido Alcázar
Eva Martínez Ortigosa

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, hoy de 2016

Controlador Neuronal

Esquemas de aprendizaje de modelos internos de brazo robótico con múltiples articulaciones

Autor
Julio Jesús Vizcaíno Molina

Directores
Jesús Garrido Alcázar
Eva Martínez Ortigosa

Controlador Neuronal: Esquemas de aprendizaje de modelos internos de brazo robótico con múltiples articulaciones

Julio Jesús Vizcaíno Molina

Palabras clave: robot_controller, neural_networks, deep_learning, baxter_robot

Resumen

Calcular el modelo dinámico de un brazo robótico es la mejor manera de entender y diseñar un controlador para dicho brazo. Sin embargo, es una tarea complicada que se basa en el conocimiento del funcionamiento del robot, así como de sus características. El controlador que aquí se presenta resuelve estos problemas. Se basa en el auto-aprendizaje del modelo, así como de los parámetros del robot. Para ello hace uso de técnicas de aprendizaje automático (Deep Learning). Estas redes son sistemas de propósito general que parametrizan variables internas en la fase de entrenamiento, para así obtener un modelo concreto al final de esta fase. Este modelo es capaz de desempeñar el papel del controlador empleado para modelarlo sobre datos no visto antes.

Neural Controller: Project Subtitle

Julio Jesús Vizcaíno Molina

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Yo, **Julio Jesús Vizcaíno Molina**, alumno de la titulación Grado en Ingeniería de Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77151856n, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Julio Jesús Vizcaíno Molina

Granada a 12 de Diciembre de 2016

D. **Jesús Garrido Alcázar**, Profesor del Área de Arquitectura y Tecnología de Computadores del Departamento Arquitectura y Tecnología de Computadores de la Universidad de Granada.

D.ª **Eva Martínez Ortigosa**, Profesora del Área de Arquitectura y Tecnología de Computadores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Controlador Neuronal, Esquemas de aprendizaje de modelos internos de brazo robótico con múltiples articulaciones***, ha sido realizado bajo su supervisión por **Julio Jesús Vizcaíno Molina**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 12 de Diciembre de 2016.

Los directores:

Jesús Garrido Alcázar **Eva Martínez Ortigosa**

Agradecimientos

Poner aquí agradecimientos...

Índice general

Capítulo 1

Introducción

1.1. Motivación

1.2. Objetivos

Los objetivos del trabajo son los siguientes:

1. Estudio de los mecanismos de control implementados en el robot biomórfico Baxter.
2. Caracterización y obtención de una base de datos de movimientos utilizando los controladores incluidos en el robot.
3. Estudio de la viabilidad de implementación de un sistema de control adaptativo basado en técnicas de machine learning.

1.3. Metodología

Capítulo 2

Materiales y métodos

2.1. ROS

ROS (Robot Operative System) [1] es un sistema operativo de código abierto basado en paso de mensajes diseñado para plataformas robóticas. Sus usos van desde el diseño de drivers, a resolver problemas complejos como la navegación autónoma.

La filosofía del sistema operativo es permitir a los desarrolladores abarcar de manera independiente los problemas que surgen al desarrollar una plataforma robótica, de modo que cada módulo pueda comunicarse con el resto, y que no haya que rediseñar un programa cuando se quiere cambiar de plataforma (y que una misma plataforma se beneficie de la implementación de distintos programas).

La versión usada para trabajar con el robot Baxter [2] ha sido `Indigo`, ya que es la versión soportada por el robot.

2.1.1. Arquitectura

La arquitectura de ROS sigue un modelo distribuido, donde cada nodo realiza una tarea y los nodos pueden comunicarse entre sí. Esta comunicación lo hace mediante paso de mensajes.

Nodos

Los nodos son los programas que realizan las tareas, ejecutables que utilizan ROS para comunicarse con otros nodos. Para programarlos se usan las librerías *roscpp* (para c++) o *rospy* (para Python [3]). En el presente trabajo se utiliza la implementación en Python (por motivos de compatibilidad con la interfaz de programación de aplicaciones (API) de Baxter, así como con las librerías *tensorflow* [4] y *keras* [5] explicadas más adelante).

Temas

Los temas son los canales de comunicación que utilizan los nodos para comunicarse con otros nodos. Son los "puertos" que cada tema pone hace público para permitir la comunicación con otros nodos.

Los nodos publican y se suscriben a los temas.

Mensajes

La comunicación entre nodos se hace con el paso de mensajes. Estos son estructuras de datos que permiten a los nodos especificar y conocer la información que envían y reciben.

Servicios

Se trata de otra manera de comunicación, en la que un nodo hace una petición a otro y éste puede responder. Se diferencia de los mensajes en que el suscriptor puede responder al mensaje recibido, y su uso está enfocado a tareas aperiódicas.

Parámetros

Los parámetros son piezas de información contenidas en cada nodo que pueden ser vistas y modificadas por los nodos.

Ejemplo

Como ejemplo, se puede pensar en una cámara de vídeo que retransmite el vídeo en tiempo real. Esta cámara es un nodo (`/camera`) que publica mensajes del tipo imagen (que consiste en una matriz de nxm píxeles) al tema `/camera/image`.

Por otro lado, en un servidor estamos ejecutando un programa de detección de imágenes haciendo uso de `ROS` para la comunicación (nodo `/camera/server`). Este programa se suscribe al tema `/image` para recibir las imágenes que va a analizar.

Adicionalmente, la cámara tiene habilitado un servicio para reiniciar la cámara (`/camera/reset`) así como un conjunto de parámetros para filtrar las imágenes antes de publicarlas (`/color_mode`), que permite cambiar la imagen a blanco y negro, sepia...

2.1.2. `rosbag`

`rosbag` [6] es una herramienta para grabar el paso de mensajes realizados sobre los temas que se le ofrecen como parámetros. Tiene la capacidad de grabar información adicional a los mensajes, como la marca temporal (momento en el que el mensaje es registrado) y de secuencia (número de

mensaje en una secuencia) que cada mensaje tiene asociada. Esto es útil para sincronizar mensajes generados por otro sistema y recibidos por tcp/ip con los mensajes generados en el propio ordenador.

2.2. Baxter

El robot biomórfico Baxter se trata de un robot de bajo coste (28.000 euros) y de baja fuerza, capaz de trabajar en entornos seguros, con otras personas a su alrededor. Esto se debe al uso de motores de bajo torque, así como del uso de actuadores elásticos en serie [7].

Cuenta con un modelo de movimiento automático basado en aprendizaje de ejemplos dados por una persona, lo que permite a personal no cualificado enseñarle a realizar tareas.

2.2.1. Hardware

Estructura física

El robot cuenta con:

2 brazos Con 7 articulaciones (grados de libertad) cada uno y sensores que miden la posición, velocidad y torque aplicado en cada una de las articulaciones.

Pantalla En la cabeza, con la capacidad de moverse hacia los lados.

3 cámaras Una al final de cada brazo y una en la pantalla, con una resolución máxima de 1280 x 800 píxeles y una tasa de refresco de 30 cuadros por segundo.

Pinzas Que se colocan al final de cada articulación. También se puede colocar una bomba de vacío. No interfieren con las cámaras.

Sonar Ubicado en la cabeza.

Botones Para interactuar con el robot, ubicados en los brazos, así como en el torso, a cada lado del robot.

Las articulaciones son:

s0 Rotación del hombro.

s1 Traslación del brazo.

e0 Rotación del brazo.

e1 Traslación del antebrazo.

w0 Rotación del antebrazo.

w1 Traslación de la mano.

w2 Rotación de la muñeca, a fin de satisfacer dicho movimiento cuando la pinza se encuentra insertada.

Dichas articulaciones se pueden observar en la figura 2.1.

Procesamiento

En el interior del Baxter contamos con un procesador Intel Core i7-3770 a 3.4 GHz con tarjeta gráfica integrada, 4 GB de memoria RAM, y 128 GB de disco duro SSD.

Actuadores elásticos en serie

Los motores utilizados para las articulaciones utilizan actuadores elásticos en serie, en contraposición a los actuadores rígidos, usados en tareas de gran precisión.

Los actuadores rígidos basan su correcto funcionamiento en la cantidad de torque que pueden aplicar: a mayor torque, mayor precisión. Suelen ser controlados sin realimentación, y basan su control en un mapeo de voltajes y posiciones. Al tener tanto torque los motores, cualquier fuerza externa se vuelve despreciable, por lo que se puede confiar su control a uno basado en el conocimiento del motor.

Es este gran torque el que hace que los robots diseñados con estos motores no sean seguros para las personas. Los robots industriales en cadenas de montaje usan este tipo de actuadores.

Por otro lado, los actuadores elásticos basan su funcionamiento en la cantidad de torque que aplican, lo que significa que un controlador basado en realimentación es necesario, ya que el motor no conoce la posición en la que está, sino el torque que está aplicando. Esto lo consigue haciendo uso de elementos elásticos, los cuales aplican una fuerza sobre el objeto a mover. La posición de dicho objeto se mide y se transmite al controlador, que aplicará la fuerza correspondiente.

Los robots basados en estos actuadores son más seguros, ya que la cantidad de torque se puede controlar, ya sea por software o por la propia construcción hardware. Baxter utiliza este tipo de actuadores.

2.2.2. Disposición

Para trabajar con el robot se dispone de un laboratorio de dimensiones reducidas, teniendo en cuenta el rango de movimiento del robot y la compartición del espacio con más gente.

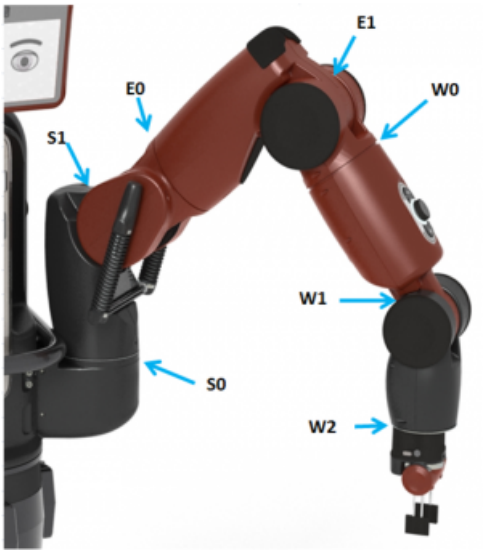


Figura 2.1: Articulaciones del robot Baxter

Además del robot, es necesario un equipo de trabajo, que consistirá en un ordenador de sobremesa con un procesador Intel Core i7-5930k a 3.5 GHz, 32 GB de memoria RAM y 200 GB de disco SSD y 1 TB de disco HDD.

Este equipo se usará para comunicarse con el robot, ejecutando el programa en el ordenador y transmitiendo mensajes entre el robot y éste haciendo uso de `ROS` .

2.2.3. Modos de control

Cada brazo se puede controlar de manera independiente, pudiendo usar cada uno de ellos con un modo de control distinto. Los modos de control son los que se muestran a continuación:

1. Control por posición
2. Control por velocidad
3. Control por torque
4. Control por posición sin procesar (raw mode)

Haciendo uso de la arquitectura de paso de mensajes que nos ofrece `ROS` , somos capaces de enviar en un tema (topic) tanto el modo de control que queremos usar como las características objetivos que queremos en el movimiento. El robot Baxter estará escuchando cada uno de los mensajes que le enviemos a los temas `/robot/limb/left/joint_command` para el brazo

izquierdo y `/robot/limb/right/joint_command` para el brazo derecho. El tipo de mensaje que se envía es `JointCommand`, que tiene como argumentos:

- mode (POSITION_MODE=1, VELOCITY_MODE=2, TORQUE_MODE=3, RAW_POSITION_MODE=4)
- command (lista de flotantes)
- names (lista de nombre de articulaciones)

El modo de control para cada brazo es único, mientras que la posición/-velocidad/torque deseado es único para cada articulación en cada brazo.

Aparte de enviar este tipo de mensajes, podemos definir la velocidad relativa máxima que alcanzará cada articulación en el modo de control por posición publicando al tema `/robot/limb/<right/left>/set_speed_ratio` un valor entre 0 y 1.

El robot Baxter cuenta con una serie de limitaciones, en cuanto a posiciones, velocidades y torques se refiere, para cada articulación. Estas limitaciones se muestran en el cuadro 2.1.

2.2.4. API de `ROS`

Baxter hace uso de `ROS` para comunicarse internamente, así como para recibir y enviar información con el exterior.

La interfaz de programación de aplicaciones (API) de Baxter con respecto a `ROS` pone a disposición del programador los tipos de mensajes y servicios usados por el mismo, así como de los temas publicados para su control.

En este trabajo se hace uso de los siguientes temas:

- `/robot/joint_states` Publica información tanto de la posición como de la velocidad y torque de las articulaciones de todas las articulaciones.

Cuadro 2.1: Límites articulaciones

Art.	(rad) Mín	(rad) Máx	(rad) Rango	(rad/s) Vel máx	(Nm/rad)
S0	-1.7016	+1.7016	3.4033	2.0	843
S1	-2.147	+1.047	3.194	2.0	843
E0	-3.0541	+3.0541	6.1083	2.0	843
E1	-0.05	+2.618	2.67	2.0	843
W0	-3.059	+3.059	6.117	4.0	250
W1	-1.5707	+2.094	3.6647	4.0	250
W2	-3.059	+3.059	6.117	4.0	250

- `/robot/limb/left/set_speed_ratio` Tema al que publicar para ajustar la ratio de velocidad usada por cada articulación del brazo izquierdo (única para todas).
- `/robot/limb/left/joint_command` Tema al que publicar para mover el brazo a las posiciones deseadas del brazo izquierdo (una para cada articulación).

2.2.5. API de Python

La API tiene como objetivo disponer una interfaz basada en el lenguaje de programación Python para controlar y monitorizar el robot Baxter, ejecutando en su base las correspondientes instrucciones `ROS` . Para ello, cuenta con una serie de módulos orientados a los distintos componentes del robot (brazos, pinzas, cámara...).

Se hará uso del módulo `brazo` para realizar el movimiento del mismo. Por motivos de disposición del robot en el laboratorio, se extraerá la base de datos haciendo uso del brazo izquierdo.

Dentro de este módulo, se hará uso de las funciones `set_joint_position_speed(speed)` y `move_to_joint_positions(positions)`. La primera controlará la velocidad máxima relativa para cada articulación, mientras que la segunda moverá el brazo a la posición deseada. Esta función cuenta además con un intervalo de tiempo máximo para realizar el movimiento. Que el tiempo máximo se agote será indicador de que el robot no puede alcanzar la posición deseada (está colisionando consigo mismo).

La función `move_to_joint_positions(positions)` realiza un filtro paso baja de la diferencia de posiciones (actual y deseada) en el tiempo, ofreciendo un movimiento más fluido.

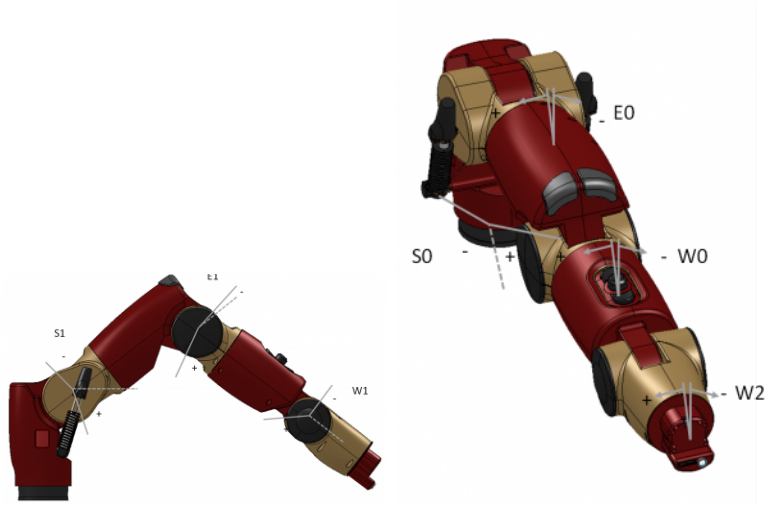
2.2.6. Mecanismos de control

Control por posición

El control por posición consiste en alcanzar las posiciones objetivo para cada una de las articulaciones. El modo de control en el mensaje JointCommand es el 1.

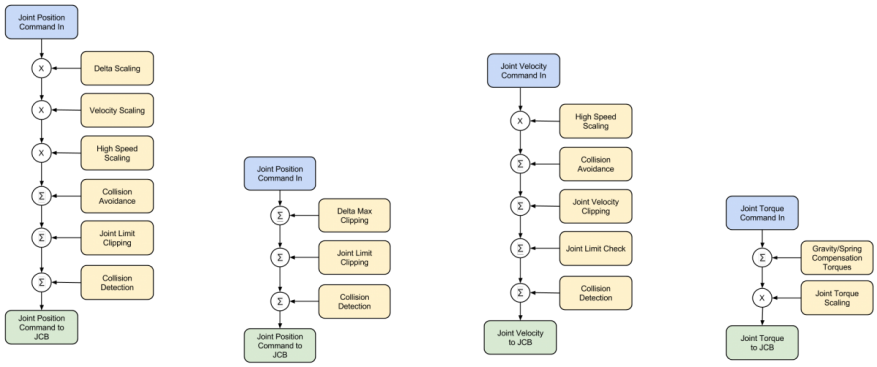
La orden donde todas las articulaciones se ponen en la posición 0 corresponde con el brazo totalmente estirado y con el hombro, codo y muñeca mirando hacia abajo. Las traslaciones y rotaciones se corresponden con los de la figura 2.2.

Dada la naturaleza del robot Baxter, en este modo de operación se aplican unos filtros antes de aplicar la orden de posición, a fin de evitar accidentes y otorgar una experiencia de movimiento más fluida y segura. Los filtros son los que se muestran en la figura 2.3a.



(a) Límites traslación (b) Límites rotación

Figura 2.2: Límites articulaciones



(a) Control por posición (b) Control por posición sin procesar (c) Control por velocidad (d) Control por torque

Figura 2.3: Filtros aplicados

1. Escalado delta. Consiste en escalar las posiciones objetivo en el recorrido para conseguir que todas las articulaciones lleguen al punto deseado a la vez.

2. Escalado de velocidad. Se escala la velocidad de cada articulación en función del 'ratio de velocidad'.

3. Escalado de alta velocidad. Para evitar colisiones entre los dos brazos moviéndose el uno en la dirección del otro, se escala la velocidad y recalcula la posición cuando ésta supera un umbral.

4. Prevención de colisiones. Gracias a un modelo interno del robot, se limitan las posiciones de las articulaciones tales que provocarían un choque con el propio robot.

5. Recorte de posiciones. Si las posiciones superan los límites de la articulación, estas se recortan al límite permitido.

6. Detección de colisión. Si se detecta una colisión (con un objeto externo al robot), mantiene la última posición antes de la colisión.

Como se puede observar, Baxter aplica unos filtros para evitar y detectar colisiones. Esta tarea se realiza de tres maneras:

- Prevención. El robot ejecuta una simulación interna, donde las articulaciones y el cuerpo cuentan con unas regiones de seguridad que, al tocarse, limitan el movimiento del robot.

- Detección de colisión. Esto se realiza de dos maneras

Impacto Cuando el torque de cualquier articulación cambia bruscamente, se considera que ha habido una colisión.

Retención Cuando el torque aplicado aumenta pero la articulación no se mueve, se considera que está colisionando con un objeto inmóvil.

- Escalado de alta velocidad. Cuando la velocidad del brazo supera los 0.2 m/s, las regiones de seguridad que el robot simula internamente aumentan. Cuando se tocan estas regiones, la velocidad se escala y la posición se recalcula.

Control por posición sin procesar

Este modo de control, al igual que el anterior, tiene como objetivo alcanzar la posición deseada para cada articulación. Se diferencia en los filtros que aplica (figura 2.3b).

1. Recorte de delta máximo. Recorta la posición siguiente (en el intervalo de actuación de cada articulación) a la máxima posición alcanzable a la velocidad máxima dada por la 'ratio de velocidad'.

2. Recorte de posiciones. Al igual que en el modo de control anterior, si las posiciones superan los límites de la articulación, estas se recortan al límite permitido.

3. Detección de colisión. Cumple la misma función que en el modo de control por posición.

Como se puede observar, este es un modo de control más avanzado, ya que no tiene en cuenta las colisiones consigo mismo, y ofrece un movimiento más brusco al no llegar todas las articulaciones al punto destino a la vez.

El modo de control en el mensaje JointCommand es el 4.

Control por velocidad

En este modo de control, lo que se busca es adquirir la velocidad objetivo para cada articulación. El modo de control en el mensaje JointCommand es el 2. Al igual que en los modos anteriores, se aplican una serie de filtros (figura 2.3c).

1. Escalado de alta velocidad. Como en el control por posición.

2. Prevención de colisiones. Como en el control por posición.

3. Recorte de velocidades. Se limitan las velocidades para cada articulación para que no excedan el máximo permitido.

4. Comprobación de límites. Se comprueban los límites de las articulaciones. Si se exceden, se detiene el movimiento.

5. Detección de colisión. Igual que en el control por posición.

El dejar de enviar velocidades si se exceden los límites de las articulaciones es una medida de seguridad que exige al programador tener el control sobre las posiciones que puede alcanzar el brazo.

Control por torque

El control por torque consiste en el modo de más bajo nivel que se puede controlar el robot Baxter. Los torques que enviemos serán aplicados por los controladores, haciendo uso solamente de los siguientes filtros (figura 2.3d):

1. Compensación. Los torques se suman a los necesarios para mantener la gravedad 0 y el muelle ubicado en la articulación s1 (traslación del hombro).

2. Escalado de torque. Si un torque excede el torque máximo para esa articulación, escala los torques de todas las articulaciones por el factor de exceso de esa articulación (torque_max/torque).

La compensación de la gravedad se puede desactivar mandando un mensaje vacío al tema `/robot/limb/right/suppress_gravity_compensation` para el brazo derecho, o `/robot/limb/left/suppress_gravity_compensation` para el brazo izquierdo.

De esta manera, un torque aplicado de 0 en todas las articulaciones, mantendrá el brazo en un estado de gravedad 0 si la compensación esta activa. De no estarlo, el brazo caerá en peso muerto.

2.2.7. Simuladores

V-REP

Virtual Robot Experimentation Platform (V-REP) [8] es un simulador robótico muy potente, con una gran cantidad de modelos predefinidos listos para simular, y una gran cantidad de opciones para configurar.

Gazebo

Al igual que V-REP, Gazebo [9] es un simulador de plataformas robóticas. Es menos potente, y tiene menos modelos predefinidos, pero está integrado con `ROS` , y el robot Baxter cuenta con un modelo listo para simular para el mismo, por lo que es la opción elegida para realizar las pruebas.

Problemas La versión del simulador en la que está implementada es la 2.2, mientras que la que se puede descargar ahora es la 7.1. Esto hace que no sea lo potente que debiera, y a la hora de realizar simulaciones utilizando el control por torques, el simulador no responde igual que el robot, por lo que su uso quedó limitado a las primeras pruebas que se hicieron en una primera aproximación al control por posición del robot.

2.3. Redes Neuronales

Una red neuronal artificial [10] [11] [12] es un conjunto de algoritmos de aprendizaje automático capaces de extraer modelos a partir de un conjunto de datos de aprendizaje. De esta manera, estos sistemas son capaces de emular la fuente generadora de datos y producir salidas coherentes a partir de entradas no vistas con anterioridad.

2.3.1. Topología

En función de la topología de la red, se contemplan dos tipos fundamentales: las redes hacia adelante, y las redes recursivas. A continuación se habla de las redes hacia delante, mientras que las redes recursivas se analizan en un apartado más adelante.

Redes hacia adelante

Las conexiones entre las neuronas es de un solo sentido, de modo que no se forman bucles entre ninguna de las neuronas (figura 2.4). Las líneas que conectan las "neuronas" son pesos, y la interconexión entre una capa y otra se puede representar como una matriz de mxn, donde m es el número de neuronas de la capa anterior, y n es el numero de neuronas de la capa siguiente. De este modo, la entrada de cada capa no es sino la combinación lineal de las salidas de las capas anteriores.

Si las relaciones entre las capas sólo fueran combinaciones lineales, la red neuronal se podría reducir a una única matriz que interconectara la capa de entrada con la de salida. Lo que sucede es que cada neurona tiene una activación, que consistirá en una relación no lineal entre la entrada que percibe y la salida. Son estas no linealidades las que le confieren la capacidad de aprender parámetros nuevos, por lo que podrán ajustarse al espacio de búsqueda.

Activaciones Las activaciones más usadas son las siguientes.

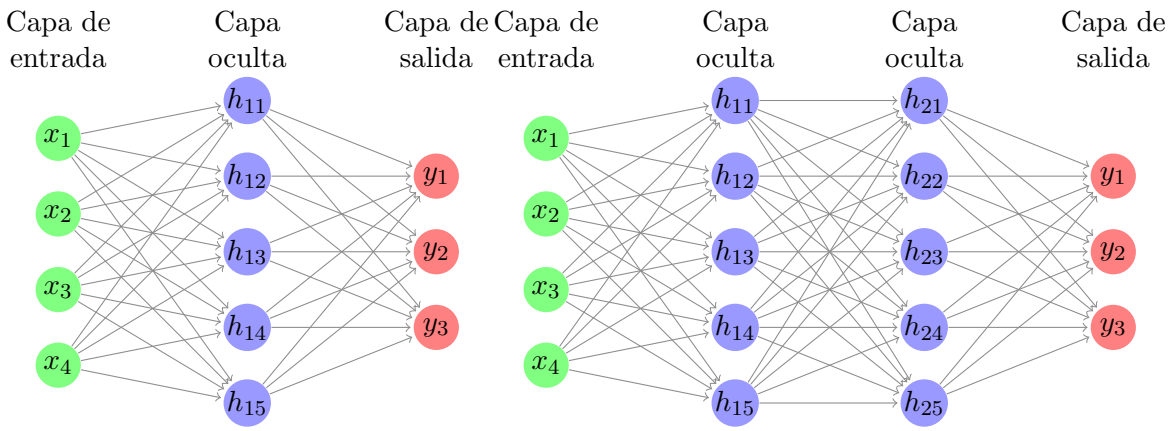
Sigmoidea Consiste en una función cuyo rango va de 0 a 1. Esto permite interpretar su salida como una probabilidad. Es muy usada en tareas de clasificación (figura 2.5a).

Sigmoidea "dura" Se trata de una aproximación de la función sigmoidea a trozos, tal y como se muestra en la figura 2.5b.

Tangente hiperbólica Tiene un comportamiento parecido a la sigmoidea, pero su rango va de -1 a 1, y es simétrica. (figura 2.5c).

Rectificador lineal (ReLU) Se trata de la función lineal rectificada. Es muy usada en redes neuronales profundas (explicadas más adelante) debido a las características de la derivada (figura 2.5d).

Errores Para calcular el rendimiento de una red se utilizan los errores a la salida, que consiste en aplicar una función a los valores obtenidos y los deseados. Las funciones más comunes son las siguientes:



(a) Red monocapa (b) Red multicapa

Figura 2.4: Redes neuronales de propagación hacia adelante

MAE El error absoluto medio (MAE) es la media del valor absoluto de la diferencia de los errores cometidos en cada neurona a la salida. Usado en problemas de regresión lineal (la salida es una curva función de los datos de entrada).

MSE El error cuadrático medio (Mean Squared Error) consiste en la media de los cuadrados de la diferencia de los errores cometidos en cada neurona a la salida. Se utiliza también en regresión lineal.

Entropía cruzada Su fórmula es $H(p,q) = -\sum_x p(x) \log q(x)$ y es usada en tareas de clasificación, ya que da un mayor sentido a la probabilidad que representa la salida de este tipo de redes.

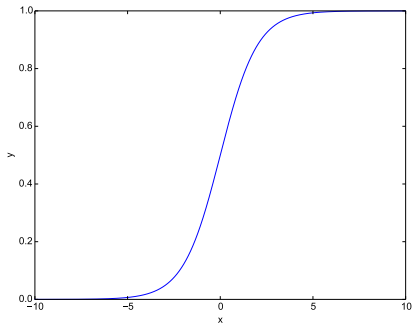
Perceptron Fue el primer tipo de red neuronal. La manera de obtener las salidas es igual a la explicada anteriormente.

El entrenamiento de este tipo de red neuronal consiste en minimizar el error de los pesos (relaciones entre las neuronas). Esto significa que, suponiendo que hay un conjunto de pesos que permite obtener la salida correcta para cada una de las entradas, el perceptrón iterará sobre cada uno de los pesos hasta encontrar un subconjunto de estos pesos.

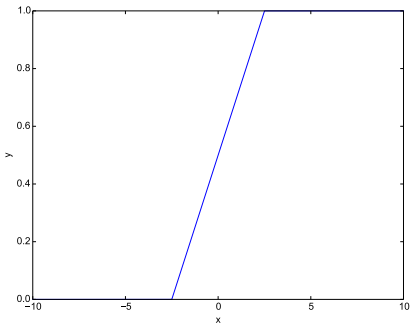
El problema radica en que puede no existir dicho conjunto, y por lo tanto el perceptrón no será capaz de encontrar los pesos.

Es por esto por lo que su uso queda restringido a problemas donde las características sean separables.

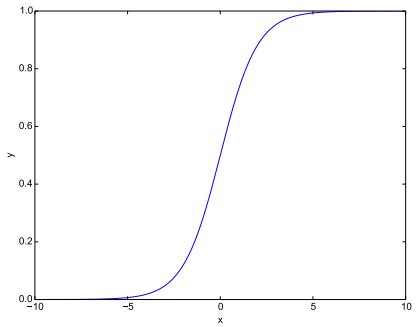
Propagación hacia atrás de errores Otra manera de entrenar la red, consisten en encontrar el conjunto de pesos que minimiza una función error,



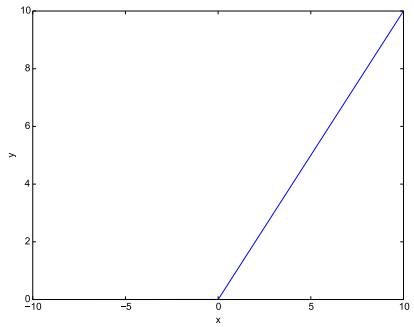
(a) Función sigmoidea



(b) Función sigmoidea “dura”



(c) Tangente hiperbólica



(d) Rectificador lineal

Figura 2.5: Activaciones

generada por los errores cometidos a la salida de la red.

Al contrario que en el perceptrón, esta tipo de entrenamiento permite encontrar un mínimo local en el espacio de los pesos para el conjunto de datos, aunque estos no sean separables.

El algoritmo de actualización de los pesos usado es el de propagación hacia atrás de errores. Consiste en, a partir del error generado en la salida, se reduce cada uno de los pesos en función de la cantidad de error asociado a dicho peso. Para realizar este algoritmo, se requiere de las derivadas de los errores cometidos en cada neurona, a fin de obtener la relación entre el error final del que cada neurona es responsable.

Hay distintos tipos de algoritmos [13] que implementan la propagación hacia atrás de errores.

Descenso en gradiente Consiste en el algoritmo más sencillo. Dada la relación entre el error total y el generado por cada peso, modifica el peso de manera proporcional a dicho error. Esto, visto en el espacio de los pesos con respecto al error total, significa descender por la curva en la dirección y sentido de mayor pendiente (de ahí el nombre).

Descenso en gradiente estocástico Es como el anterior, pero, en vez de calcular el error para todo el conjunto de datos de entrenamiento, se calcula el error en pequeños grupos y se aplica el algoritmo. Esto produce que el descenso no se produzca en la dirección de máxima pendiente del conjunto, sino del subconjunto. El método funciona porque la media de dichos descensos aproxima el descenso total. Al caso en el que el subconjunto es una sola muestra se le llama aprendizaje online.

Momentum Es una modificación al descenso en gradiente, en el que la dirección de descenso se ve alterada por los descensos producidos en iteraciones anteriores. Esto le da la característica de "velocidad" al punto de operación.

Adam La estimación adaptativa del momento hace uso de la técnica del momento, y adapta la cantidad de descenso en gradiente para cada uno de los parámetros a optimizar, de modo que, los parámetros cuyo rango de operación es más pequeño perciben cambios más pequeños en la actualización de su valor.

Redes recursivas

Las redes recursivas [14] son el otro tipo de redes en función de la topología de la red, y difieren de las redes hacia delante en que las neuronas pueden conectarse entre sí hacia atrás (o consigo mismas), generando en la

red una realimentación entre nodos que provoca que la salida de la red no dependa sólo de la entrada, sino del estado de la misma (que varía en función de su historia) (fig. 2.6).

Tienen la capacidad de aprender de sistemas cuyo comportamiento cambia con el tiempo.

Redes recurrentes Es el tipo de red recursiva más básica. Consiste en un conjunto de neuronas (normalmente agrupados en una capa) cuya salida no depende solamente de las entradas provenientes de capas anteriores, sino que también dependen de las salidas de las neuronas en esa misma capa. De esta manera, el estado de la red cambia conforme cambian las salidas de esta capa.

LSTM Las redes de memoria a corto y largo plazo (Long Short Term Memory) [16][15] hacen uso de una unidad especial, que consiste en un conjunto de puertas que permiten el flujo de información a través de la neurona. En concreto, cuenta con 3 puertas (fig. 2.7a):

- Puerta de entrada. Su valor varía de 0 a 1 y se multiplica al valor de la entrada de la activación.
- Puerta de salida. Igualmente, su valor varía de 0 a 1 y se multiplica al valor de la salida después de la activación.
- Puerta de olvido. Representa la cantidad del estado anterior que se mantiene.

Todas estas puertas son activaciones sigmoideas que se activan con las entradas de la celda y con el estado de la misma.

GRU Las celdas GRU (Gated Recurrent Unit) [17] son, al igual que las LSTM, celdas que hacen uso de puertas. Son más sencillas que las anteriores, y requieren menos cómputo, por lo que son más rápidas de entrenar. Cuentan con dos puertas en vez de tres: reinicio y actualización (fig. 2.7b)

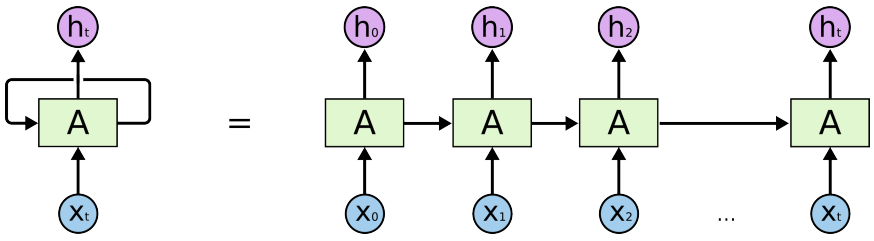


Figura 2.6: Red neuronal recurrente [Fuente: [15]]

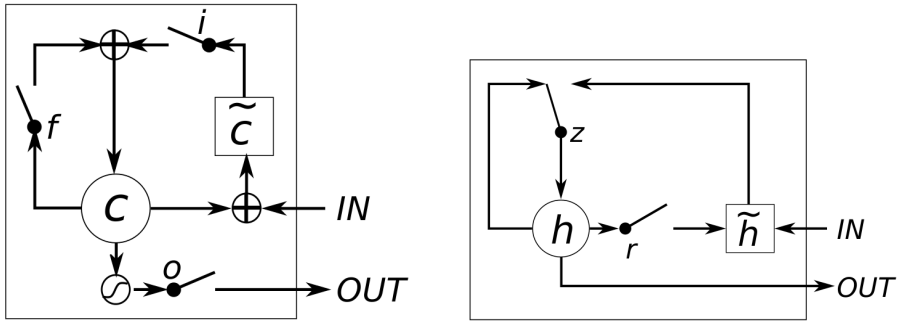
La salida es una interpolación lineal del estado anterior y el nuevo estado propuesto, el cual se calcula como una combinación de la entrada y el estado anterior (multiplicado por la puerta de reinicio). La puerta de actualización se calcula a partir del estado anterior y la entrada, al igual que la puerta de reinicio.

Propagación hacia atrás de errores en el tiempo El algoritmo de optimización de parámetros en el tiempo es el mismo usado en las redes hacia delante (propagación hacia atrás de errores), con la peculiaridad de que la red se desenrolla en el tiempo. Esto significa, que las neuronas realimentadas se interconectan entre sí en cada instante temporal (desenrollado). Las pérdidas se calculan de la misma manera, calculando el error en cada instante temporal (si es que se tiene una salida en cada instante) y aplicando el algoritmo de propagación hacia atrás sobre cada neurona.

Esta es la principal razón por la que surgieron las celdas LSTM y GRU, ya que, haciendo uso nada más de las redes recurrentes, se tiene el problema de desvanecimiento y explotación de los pesos de la red [18]. Este fenómeno surge cuando la red se desdobla un número lo suficientemente grande como para que, al propagar hacia atrás el error, debido a la gran dimensionalidad de la red, el error llegue muy reducido (desvanecimiento) o muy amplificado (explotación), impidiendo que, alcanzado este punto, la red aprenda.

2.3.2. Regularización

Las redes neuronales son sistemas capaces de aprender a extraer patrones de los datos que se le ofrecen. Por lo tanto, se podría pensar que una red lo suficientemente grande es capaz de extraer cualquier patrón que haya en los datos. Esto es ideal si contamos con una base de datos infinita, ya que cualquier discrepancia entre muestras se tomaría como ruido y la red no



(a) Celda LSTM [Fuente: [17]] (b) Celda GRU. [Fuente: [17]]

Figura 2.7: Celdas usadas en redes recursivas

aprendería de éste. El problema es que no se puede aprender de una base de datos infinita (el tiempo de cálculo sería igualmente infinito), y no se puede obtener dicha base de datos.

Una vez asumido que no se puede disponer de esta base de datos, nos encontramos con el problema de que, en una base finita, podemos no tener representados todos los casos de estudio que nos interesan, esto es, que la base de datos no sea representativa de la realidad de la que se pretende aprender. En este caso, lo mejor que puede hacer la red es aprender de dichas muestras, y sería incapaz de generalizar para el conjunto no visto (o no visto lo suficiente).

También nos encontramos con que cada muestra por sí misma puede contener ruido (de los sensores, de un error en el proceso de obtención...). En este caso, si no se toman medidas, la red aprenderá también estos patrones, lo que reducirá aumentará el error cuando se enfrente a muestras que no ha visto antes (y sobre las que el ruido ha influido de manera distinta).

Es por esto, por lo que se requieren de técnicas de regularización, para evitar el sobreentrenamiento de la red (overfitting). Algunas de estas técnicas son las siguientes.

Infradimensionamiento de la red

La primera medida que se puede tomar es ajustar el tamaño de la red al problema que se esté abarcando. De esta manera, se elimina el problema de base de que las redes más grandes son las que mejor encuentran patrones (en este caso de estudio, de ruido).

L2

El problema del sobredimensionamiento es que las neuronas se especializan demasiado (capaces de reconocer el ruido). Esta especialización se da debido a que algunos pesos de interconexión de neuronas se vuelven muy altos, eliminando la influencia que tienen el resto de entradas en una misma neurona.

Una solución a este problema es añadir un error proporcional al cuadrado de los pesos de cada neurona (error L2). De esta manera, los pesos altos (especializados) aumentarán el error, y en el paso de propagación hacia atrás de errores se reducirá dicho peso.

L1

La regularización L1, al igual que la L2 [19], consiste en la aportación de los pesos al error, pero a diferencia de ésta, esta aportación es lineal proporcional (no cuadrática). Esto ayuda a seleccionar un conjunto de neuronas,

que se especializarán, pero, al ser un número reducido de ellas, esta especialización promoverá el aprendizaje de los patrones más importantes de los datos de entrenamiento, y por lo tanto, la eliminación del ruido.

Dropout

Dropout [20] es una técnica de regularización que se basa en el hecho de que, la media ponderada de un conjunto de redes neuronales, cada una especializada en un subconjunto de datos, genera unos resultados mejores que cada una de las redes individualmente. Esto se debe a que cada red puede haber adquirido un error distinto, pero al juntarlas, este error disminuye.

El problema es que entrenar redes neuronales es un proceso computacionalmente costoso. Es por esto por lo que esta técnica propone realizar dicha media en la fase de entrenamiento. Para ello, en la fase de propagación hacia adelante, anula cada neurona con una probabilidad p, de modo que la salida va a depender de un subconjunto de las neuronas de la red (entre las cuales pueden estar o no las neuronas especializadas). De esta manera, se consigue que las dependencias con las neuronas más especializadas se reparta sobre varias neuronas, de modo que se reduce el error, al depender de varias neuronas, cada una con su versión reducida del error.

2.3.3. Aprendizaje profundo

Aprendizaje profundo (Deep Learning) [21][22] es un conjunto de técnicas usadas sobre las redes neuronales que le permiten alcanzar resultados nunca antes conseguidos en el campo del aprendizaje automático.

En un sentido amplio, el aprendizaje profundo consiste en una red neuronal con un gran número de capas y un gran número de neuronas por capas. Esto le da la capacidad de abarcar cualquier problema, pero también tiene el problema de que la red, de no tomar las medidas adecuadas, aprenderá incluso el error que hay en la base de datos de entrenamiento (overfitting).

Este error se disminuye haciendo uso de la técnica de regularización dropout, utilizado en conjunción con el error L2 y/o el error L1.

Este tipo de redes son realmente potentes cuando se entrenan con grandes bases de datos de entrenamiento. El problema de estas bases de datos, es que para realizar el algoritmo de propagación hacia atrás de errores, es necesario calcular el error de cada neurona, y para ello hay que calcular la derivada y la aportación de cada neurona al error. Esto supone una gran cantidad de recursos (tanto de procesamiento, realizado en GPU, como de memoria) que en ocasiones no se puede abarcar. Para solucionar este problema, se utilizan algoritmos de propagación hacia atrás de errores basados en la división de la base de entrenamiento en grupos (descenso en gradiente estocástico).

Como se ha comentado, hay que calcular las derivadas para conocer la aportación de cada neurona al error. Esto supone un gran coste compu-

tacional, por lo que se utilizan funciones no lineales con derivadas fáciles de calcular, como las activaciones relu o la sigmoide "dura".

2.3.4. Evaluación

Para evaluar el rendimiento de la red neuronal, se divide la base de datos en dos conjuntos, uno de entrenamiento y otro de test. El primero servirá para entrenar la red, cada uno de los pesos. El segundo servirá para calcular el error que comete esa red sobre un conjunto de datos no vistos en el entrenamiento.

Además, se puede dividir los datos en un tercer conjunto, llamado de validación [23], para ajustar parámetros de la red, como número de neuronas por capa, número de capas, proporción usada en dropout, en L1, L2, distintas topologías... El motivo por el que no se puede usar el conjunto de test para este fin y ha de utilizarse uno distinto, es debido a que este conjunto lo vamos a utilizar para tomar decisiones sobre la red basadas en el rendimiento que se obtiene con dicho conjunto. Es por esto, por lo que el error obtenido en el conjunto de test (de usarse como de validación) no sería válido, ya que la red ha sido diseñada para adaptarse a dicho conjunto, y por lo tanto, a la hora de enfrentarse a datos nuevos, no desempeñaría con la eficacia esperada.

2.3.5. Herramientas

Diseñar los algoritmos mencionados no es tarea fácil, y mucho menos hacerlo con la eficiencia que se necesita para que estas técnicas sean viables. Es por esto por lo que existen diversas plataformas de diseño de redes neuronales, con los algoritmos de propagación hacia atrás de errores, las topologías, activaciones y métodos de regularización ya implementadas.

Entre ellas se encuentran *Theano*, *Tensorflow* [4], *Caffe* [24], *Torch* [25]... Además, contamos con librerías que, sobre estas anteriores, ofrecen una capa de abstracción para diseñar y entrenar redes neuronales profundas, como Keras [5] y Lasagne [26].

Las librerías usadas en este trabajo son *Tensorflow* y *Keras*, debido a que están escritos en Python (facilitando la integración con la API de Baxter, así como *rospy*) y que cuentan con una gran comunidad detrás que hace sencillo su aprendizaje y la resolución de problemas que van surgiendo conforme se va desarrollando.

Tensorflow

Creada por Google, esta librería se basa en la descripción de la red mediante variables simbólicas. El conjunto de estas variables y las relaciones entre ellas se denominan gráficos. Una vez creados los gráficos, se ejecutan en el contexto de sesiones, ofreciendo así una forma de abstracción con respecto

a lo que sucede en los gráficos. Estas sesiones pueden ejecutarse en CPUs, o más eficientemente, en GPUs.

Las gráficas se forman relacionando nodos mediante operaciones con tensores. Un tensor es un vector n-dimensional. Los nodos pueden ser de entrada, constantes, variables, o módulos que implementan operaciones más complejas ya creados, como `matmul` (multiplicación de tensores), o `GradientDescentOptimizer`.

A continuación, la gráfica se carga en una sesión, se le ofrecen los valores que necesita (entradas, valores auxiliares) y se le piden los nodos que se necesitan (el de entrenamiento, para que ejecute un paso de descenso en gradiente). Las gráficas, al igual que los datos necesarios para su ejecución, se cargan completamente en la CPU/GPU, de modo que no son necesarias hacer llamadas al sistema para adquirir los recursos, haciendo la computación mucho más rápido.

Tensorboard Dentro de *tensorflow* tenemos la utilidad *tensorboard*, que es una herramienta para visualizar el entrenamiento. En ella se puede monitorizar el error de la red en función del tiempo, así como la distribución de los pesos en cada una de las capas. Es una herramienta útil para saber si la red sufre de overfitting y las razones por las que lo hace.

Keras

Keras por su parte, es una librería que ofrece al desarrollador una serie de funciones que permiten la abstracción de las operaciones que se ejecutan por debajo (sobre *tensorflow* o *theano*). Está creada especialmente para diseñar redes profundas. Cuenta con módulos como `Dense`, que no es más que una capa totalmente conectada con la siguiente, así como `GRU`, que implementa una capa formada por celdas GRU. También incluye los métodos de regularización explicados anteriormente, así como las activaciones.

Se usa diseñando la red en el entorno de un modelo. Tiene dos manera de crearla: secuencial, y utilizando directamente los modelos. La primera permite crear de manera sencilla una red donde cada capa se conecta con la siguiente, con una capa de entrada y otra de salida. La segunda permite configuraciones más complejas, como dividir la red en varias subredes, aceptar distintas entradas, y distintas salidas.

Una vez generado el modelo, sólo hace falta compilarlo (diciéndole qué algoritmo queremos usar para la fase de optimización, así como el tipo de error usado) y ejecutar la orden `fit`, que realiza una iteración sobre todo el conjunto de entrenamiento que le hayamos dado. Esta orden permite especificar el tamaño del grupo usado para realizar cada paso en la optimización (descenso en gradiente estocástico).

2.4. Controladores

Los controladores [27][28][29][30] son sistemas que modifican un estado del que no se tiene directamente control en función de una señal de entrada. Esta modificación la hacen a través de señales que se envían a los actuadores, que sí que tienen la capacidad de cambiar el estado.

A la señal que nos indica el estado deseado se le llama señal de referencia, mientras que a la generada por el controlador se le llama de control. A la

Ejemplos de controladores son los usados en regulación de la temperatura, donde no se tiene control sobre la misma directamente, sino que a lo que se tiene acceso es al voltaje aplicado a la fuente generadora de calor. Otro ejemplo es el control de velocidad de un motor. El motor es el actuador, y se le puede aplicar más o menos voltaje, en función del cual aplica el motor va más o menos rápido. A este motor se le anexiona una rueda, con lo que el motor ahora se encuentra con una fuerza que se opone a la dirección del movimiento, provocando que la velocidad del motor baje a un mismo voltaje.

Es a este tipo de problemas a los que los controladores se enfrentan y a los que deben ofrecer una solución.

2.4.1. Error distal

El error distal hace referencia a la diferencia entre el estado deseado y el estado actual. El nombre hace referencia a que no es un error que dependa directamente del controlador.

2.4.2. Control realimentado

Un sistema de control se dice que es realimentado si la señal de control que ofrece depende del estado actual. La salida del sistema será función del estado deseado y del estado actual. En concreto, este tipo de sistemas son función del error distal, y no tienen en cuenta los valores concretos de los estados deseados y actuales. En la figura 2.8 se puede observar el diagrama de bloques de este tipo de sistemas.

Un problema al que se hace frente en el diseño de controladores realimentados (como en cualquier sistema que lo sea), es la estabilidad del sistema. Es por esto por lo que se hay que estudiar que se cumpla dicha estabilidad en el diseño y uso de este tipo de sistemas.

Controlador PID

El controlador PID [31] es un controlador lineal realimentado ampliamente usado en sistemas de control (fig. 2.9).

Se caracteriza por ser la suma de tres componentes de la señal de entrada: una parte proporcional a esta, una parte de la derivada, y otra de la integral.

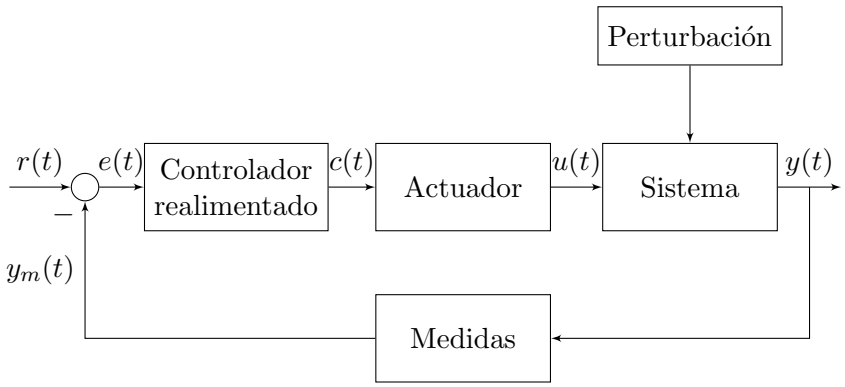


Figura 2.8: Sistema de control realimentado

De este modo, tenemos tres controladores en uno. El sistema total es la suma ponderada de cada una de estos controladores.

Controlador P Es el controlador más sencillo. Consiste en aumentar la señal de control conforme aumenta la señal de error, e igualmente, disminuirla cuando la señal de error lo hace. De esta manera, aproximamos el estado actual al deseado y disminuimos el error.

Controlador I Este controlador hace uso de la parte integral de la señal. Para ello, se tiene en cuenta la historia del sistema y se realiza una acumulación ponderada de la historia del mismo. Este controlador sirve para ajustar el error constante que puede haber entre el estado deseado y el actual dada la señal de control generada por el controlador P, una vez que este se estabiliza.

Controlador D El controlador D consiste en cambiar la señal de control proporcionalmente a la derivada de la señal de error. De esta manera, si el controlador P está acercando rápidamente el estado actual al deseado (disminuyendo así el error), este controlador generará una señal de sentido contrario a la generada por el controlador P, con el fin de disminuir las oscilaciones producidas por este cuando alcanza la posición deseada a gran velocidad.

Este controlador tiene la dificultad añadida de encontrar los parámetro que optimizan el tiempo que se tarda en alcanzar el estado deseado. Es por esto, por lo que hay técnicas como la de ciclo límite de Ziegler y Nichols [32].

Método de ciclo límite Es un método empírico para el cálculo de las constantes del PID. Se basa en que los sistemas se pueden caracterizar en función de la constante proporcional mínima que genera una oscilación en

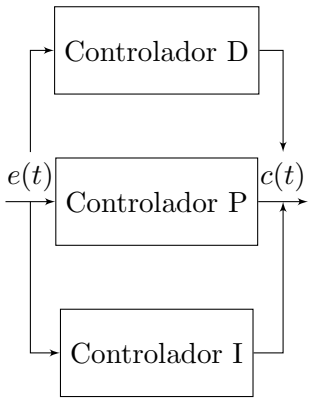


Figura 2.9: Controlador PID

el sistema mantenida en el tiempo K_u, así como con el periodo de oscilación T_u.

El método consiste en aumentar la constante proporcional (las otra se dejan a 0) hasta que se observe este comportamiento oscilatorio. Se anota la constante y el periodo de oscilación. A partir de estos valores, obtendremos las constantes según la tabla 2.2.

Ejemplo Un ejemplo de la aportación que hace cada uno de estos controladores es el de controlar la posición de un brazo robótico que opera en el plano vertical. Dada una posición deseada para cada articulación y una posición real, se obtiene una señal de error, que alimenta al controlador.

Queremos que cuanto más lejos se encuentren cada una de las articulaciones con respecto a su posición objetivo, la fuerza sea mayor (control P).

También tenemos que tener en cuenta que, una vez alcanzada la posición objetivo, las articulaciones lo harán con cierta inercia, y es aquí donde el controlador D se encarga de disminuir este efecto, ejerciendo una fuerza en sentido contrario a la velocidad en la dirección del movimiento, ya que los cambios realizados en esta dirección minimiza el error, y por tanto tienen derivada negativa.

Para terminar, habrá articulaciones afectadas por la gravedad (operan en el eje vertical), por lo que, una vez estabilizado el sistema (controlador D valdrá 0), el controlador P no generará la fuerza necesaria para combatir

Cuadro 2.2: Cálculo de constantes con el método de ciclo límite de Ziegler y Nichols

Tipo de control	T_p	T_i	T_d
PID	$0.6K_u$	$0.5T_u$	$0.125T_u$

la gravedad, que es una fuerza constante que impide el movimiento. El controlador I se encarga de ajustar esta señal, teniendo en cuenta el error en intervalos de tiempo grandes (cuando el robot ha alcanzado el equilibrio y se encuentra parado).

2.4.3. Control anticipativo

Este tipo de control (mostrado en la figura 2.10) se basa en el conocimiento del sistema que se quiere controlar. No es un sistema de lazo cerrado, y por lo tanto no tiene problemas de estabilidad.

Se trata de sistemas no lineales que describen el comportamiento del sistema que describen. Este comportamiento se basa en observaciones que se han hecho del mismo, y por lo tanto se requiere conocer la respuesta de éste a los impulsos que generan los actuadores.

Estos sistemas no reciben una señal del estado del sistema que describen, pero sí que reciben las perturbaciones que afectan al sistema, por lo que podrá anticiparse a la respuesta que tiene este a estos estímulos. Esto le permite actuar antes de que el problema ocurra, y con ello, obtener un control más rápido que el logrado uno realimentado.

Un ejemplo es el control de la posición de un motor en función del voltaje. Se puede realizar una asociación a priori entre voltajes a la entrada y posiciones a las que se llega. Si el modelo es lo suficientemente preciso, en ausencia de una fuente externa no esperada que interfiera en los resultados, se alcanzará la posición objetivo.

El inconveniente de este sistema es que la eficacia del mismo depende de lo bien que se caracterice, por lo que un análisis en profundidad del sistema es necesario (lo cual suele ser complicado). Otro inconveniente es que no obtiene una medida de lo bien o lo mal que lo está haciendo, por lo que, en presencia de mecanismos externos al sistema, el controlador no funcionará de la manera esperada.

2.4.4. Control mixto

Un modelo mixto que combine ambos tipos de controladores (figura 2.11) obtiene los beneficios de usar cada uno de ellos por separado.

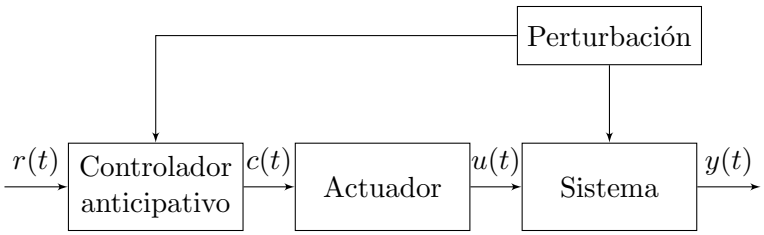


Figura 2.10: Sistema de control anticipativo

Por un lado, con el controlador anticipativo, se podrá actuar más rápido ante las perturbaciones que afectan al sistema. Por otro, se podrá corregir el ruido y los agentes externos no analizados (o no analizables) y añadidos al control anticipativo.

El funcionamiento es el siguiente. En primer lugar, actua el controlador realimentado (normalmente un PID), calculando la señal de control necesaria para alcanzar la posición objetivo. Por otro lado, el controlador anticipativo obtiene una muestra de la perturbación que afecta al sistema, y genera una señal que rectifica a la generada por el controlador realimentado. El resultado de esta señal se alimenta al actuador, que modifica el sistema. Esto se realiza en cada instante de tiempo, hasta alcanzar el estado deseado.

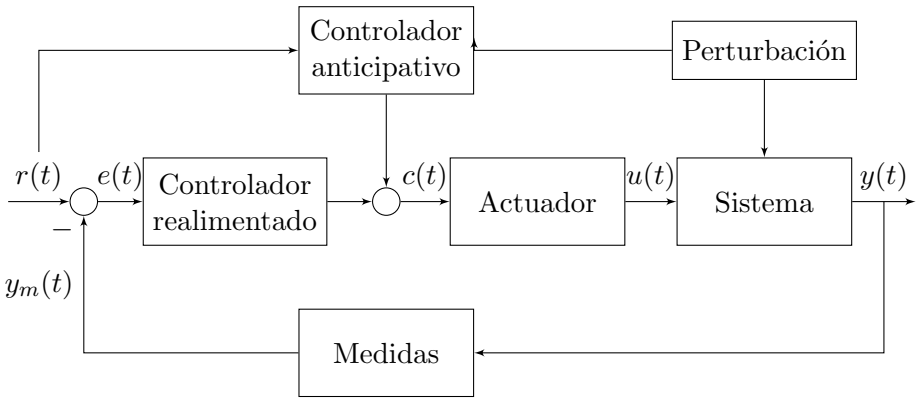


Figura 2.11: Sistema de control mixto

Capítulo 3

Diseño experimental y resultados

En este capítulo se explicarán las decisiones tomadas en cuanto al diseño del experimento con el fin de alcanzar los objetivos previamente mencionados, así como los resultados obtenidos de dicho experimento.

3.1. Trabajo previo

En un principio, se quería extraer una base de datos de torques aplicados sobre una sola articulación, manteniendo el resto en una posición fija. A partir de esta base de datos, se diseñaría y entrenaría una red neuronal que aprendiera las posiciones alcanzadas al aplicar una serie de torques durante un intervalo de tiempo. Esta red sería capaz de generar la secuencia de torques a partir de la posición objetivo y la actual, de acuerdo a los datos extraídos en el experimento.

Para ello, en primer lugar se estudiaron las herramientas necesarias para llevar a cabo el proyecto. Estas son: `ROS` , la API de Baxter y en un principio, el simulador robótico V-REP. Primero se aprendió a usar V-REP, realizando el tutorial que aparece en su manual. Con él se aprendió a realizar un robot con dos ruedas y un sensor de proximidad, capaz de moverse de manera autónoma.

Paralelamente, se estudió `ROS` haciendo uso del tutorial que ofrece la página web, en el que se aprendió los conceptos de nodo, mensaje, y tema entre otros, así como aprender a usarlos haciendo uso del terminal de linux (bash) y de la librería *rospy*. Se requirió aprender a programar en Python, mientras se aprendían estas otras herramientas.

Una vez estudiados estos temas por separado, se intentó utilizar lo aprendido de `ROS` sobre la simulación de Baxter que V-REP trae implementada. Desafortunadamente, se descubrió que el simulador no se integraba con `ROS` , por lo que se estudiaron otros simuladores que sí fueran compatibles.

Se encontró que el propio robot tenía un modelo en el simulador Gazebo, totalmente compatible con los nodos y temas que ejecuta el robot de verdad, y al cual te conectas de la misma manera que lo harías con éste. Por lo tanto, se dedicó un tiempo a estudiar el simulador, aunque ya estaba preparado para probar directamente los experimentos sobre él.

Para familiarizarse con las APIs de Baxter, se realizaron algunos programas que publicaban mensajes a los brazos del robot para controlarlos por posición, velocidad, y torque. También se experimentó con la API de Python y se descubrió en ella una manera mucho más cómoda de comunicarse con el robot.

En las pruebas realizadas, se notó un buen comportamiento del simulador en los controles por posición y velocidad, no así en el control por torques, donde el robot simulado se comportaba sin sentido. Se hizo una versión del controlador PD (sin la componente integradora) que no se comportaba como debía. Dado el comportamiento anómalo del simulador, se decidió probar el programa en el Baxter, descubriendo así el mal funcionamiento del simulador en cuanto a control por torque se refiere.

Una vez familiarizados con el robot y su control, se decide crear un programa que mueva una articulación controlada por torques, mientras que las demás se quedarían quietas (controladas por posición). Desafortunadamente, se descubre que el robot no es capaz de ejecutar este tipo de control híbrido, por lo que es necesario hacer un control por torques del brazo completo. Esto supuso realizar un controlador para las articulaciones que no se mueven. El controlador elegido fue el PID, y tras ajustar los parámetros del mismo haciendo uso del método del ciclo límite, se probó la configuración en el robot.

El controlador no funcionaba como se esperaba, produciendo muchas micro-oscilaciones y oponiendo poca fuerza al intentar mover el brazo, lo que supondría obtener muestras muy ruidosas de la articulación de la cual se deseaba obtener una base de datos. A partir de las constantes anteriores, se intentaron ajustar los parámetro del PID a mano, haciendo uso de un paquete de `ROS` que implementa el controlador PID y cuyos parámetros se pueden cambiar sobre la marcha. Los resultados mejoraron, pero seguían sin ser satisfactorios.

Se estudiaron técnicas de inteligencia artificial de ajuste de parámetros [33][34], como colonias de hormigas, recocido simulado, o algoritmos genéticos. Se implementó el algoritmo de recocido simulado sobre el PID, calculando el error del mismo como el error absoluto medio ponderado en el tiempo de la posición deseada y las recogidas. Pero las muestras eran muy ruidosas, y una misma configuración de parámetros devolvían errores muy distintos, además que, debido a las grandes fuerzas que aplicaba el robot, éste se paraba durante unos minutos, haciendo inviable el aprendizaje con este tipo de algoritmos, que requiere de miles de iteraciones.

Es por esto por lo que se reestructuró el diseño del problema. Si no se era

capaz de encontrar un controlador que mantuviese las articulaciones quietas para realizar el experimento, para así tomar muestras de los torques aplicados sobre una articulación y las posiciones a las que llegaba, se aprendería de todos los torques a la vez.

Una alternativa era realizar el mismo experimento que se quería hacer con una articulación, pero con todas a la vez. El problema era que las fuerzas aplicadas moverían el robot de manera que a priori no se puede predecir su movimiento, lo que hacía que fuera peligroso tanto para las personas que trabajaban con él como para el robot mismo. Además, obtener una base de datos de posiciones a las que llega el robot habiendo aplicados ciertos torques se vuelve infinita, ya que a una misma posición se puede llegar de distintas maneras (desde un mismo sitio) a través de distintos recorridos.

Desechada la idea de controlar por torques el robot para obtener una base de datos, se llega a la conclusión de que la manera más segura y viable, es obtener la base de datos a través del control por posición del propio robot. De esta manera, moveremos el brazo de un lugar a otro y registraremos las posiciones por las que pasa, a la velocidad que lo hace, y los torques aplicados para realizar dicho movimiento.

Se estudia realizar un muestreo de todas las dimensiones del espacio. Se estudia el caso en que a cada dimensión le asignamos dos puntos (los dos extremos de su rango). A continuación, se calcula el número de combinaciones entre todos los puntos de las 7 articulaciones. El resultado es 16256 combinaciones posibles. Suponiendo que cada movimiento tarda de media 2 segundos, y que nunca se realiza el mismo movimiento dos veces, se tardan más de 9 horas en recoger esta base de datos, que solo contiene información para realizar movimientos que van desde los extremos de las articulaciones, a otros extremos. Para tres puntos en vez de dos, el número de combinaciones posibles asciende a 4780782, que supongamos 1 segundo por movimiento (los puntos están más cerca entre sí), son 55 días y 8 horas de toma de datos. Esto sin tener en cuenta que, por motivos de seguridad, no se puede dejar el robot funcionando sin supervisión. La fórmula hallada es la siguiente:

$$combinaciones = 2 * \sum_{i=1}^{dim^n} (dim^n - i)$$

Donde dim es el número de dimensiones (articulaciones) y n el número de puntos por articulación.

Es por esto por lo que se optó por un enfoque probabilístico, donde las posiciones siguientes a las que se llegaría serían elegidas al azar, eligiendo de manera uniforma las posiciones siguientes para cada una de las articulaciones dentro del rango de cada una, tal y como se explica más adelante.

3.2. Diseño experimental

Una vez estudiados los mecanismos de control del Baxter, se procederá a extraer una base de datos del control por posición, así como diseñar una red neuronal capaz de aprender de este controlador.

3.2.1. Extracción de base de datos

La base de datos consistirá en las posiciones, velocidades y torques registradas por el robot, así como de las órdenes enviadas sobre la posición y la ratio de velocidad deseadas.

Para ello, haremos uso de la interfaz para el lenguaje Python que ofrece Baxter, así como de la herramienta `rosbag`.

Ejecución

El espacio vectorial de las siete dimensiones formadas por cada una de las articulaciones se muestreará de manera uniforme, a fin de obtener una base de datos representativa del controlador que estamos aprendiendo. Esto significa que, para cada movimiento, se elegirán siete posiciones objetivo (una por articulación) de acuerdo a una distribución uniforme sobre el rango de cada articulación. De igual manera se elegirá una ratio de velocidad objetivo con distribución uniforme entre 0 y 1. El tiempo máximo empleado para cada movimiento será de 15 segundos, tiempo suficiente para permitir el movimiento entre puntos distantes entre sí a una velocidad baja, y por lo tanto, para todos los movimientos.

Tratamiento

Una vez obtenida la base de datos, se prepara para la fase de entrenamiento y evaluación de la red neuronal.

Ficheros `.bag` El primer paso consiste en transformar la base de datos en tipos de datos que entienda el lenguaje de programación Python.

Los ficheros obtenidos con `rosbag` tienen la extensión `.bag`, y consisten en ficheros conteniendo mensajes sobre los que se iteran. De esta manera, iteramos sobre cada mensaje y leemos su contenido, que consiste en el tema del cual proviene el mensaje, una marca temporal de su recepción por `rosbag`, y del mensaje en sí mismo.

La marca temporal corresponde al tiempo del reloj interno del ordenador en el cual se recibe el mensaje. Dado que la conexión entre el robot y el ordenador se realiza por tcp/ip, este tiempo diferirá del de generación en el robot. Para paliar con este problema, `ROS` dispone una marca temporal dentro del cuerpo del mensaje con el instante de creación del mensaje en el

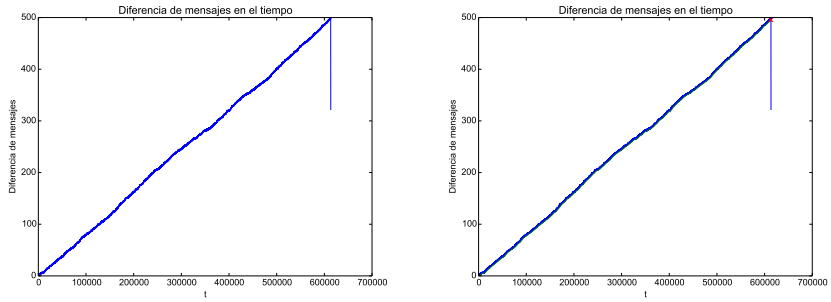
robot (de acuerdo a su reloj interno), además de un número de secuencia para ordenar los mensajes en el ordenador.

Error en frecuencia Gracias a esta información, se percibió un error en frecuencia entre el reloj del ordenador y el del robot. Esto significa que, a una frecuencia de generación de mensajes de 100 Hz, el ordenador generó 100 posiciones y velocidades deseadas por segundo (suponiendo que es el que tiene el reloj ajustado correctamente), mientras que el robot generó 99 posiciones, velocidades y torques actuales. Para solventar este problema, en primera instancia se representó gráficamente la diferencia de mensajes obtenidos por parte del ordenador y el robot en función del tiempo (figura 3.1a).

Como se puede observar, esta diferencia crece con el tiempo, lo que significa que el ordenador genera mensajes a mayor velocidad que el robot. Es cuando el ordenador deja de generar mensajes cuando esta diferencia cae en picado, momento en el cual, los mensajes generados por el robot dejarán de corresponderse con las posiciones deseadas y pasarán a corresponderse con las posiciones, velocidades y torques del robot manteniendo la última posición alcanzada.

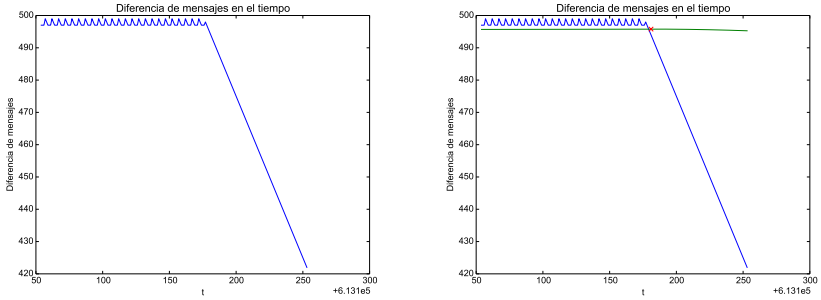
Por lo tanto, el objetivo será encontrar el momento en el que ocurre dicha bajada, es decir, el pico. El problema surge cuando la señal no es monótona, sino que, como se observa en la figura 3.2a, crece y decrece en intervalos temporales pequeños (debido a la naturaleza a ráfagas de la señal). Para paliar con este inconveniente, se realiza un filtrado paso baja (con un filtro de media móvil) que elimine dicha componente frecuencial. El resultado es el que se observa en las figuras 3.1b y 3.2b. A partir de esta señal, solo queda seleccionar el valor máximo de la misma. El valor obtenido es el número de mensaje recibido hasta el cual los mensajes son válidos.

Dado que el control lo vamos a realizar desde el ordenador, adaptamos los mensajes recibidos al mismo. Esto lo hacemos mediante interpolación



(a) Señal sin procesar (b) Señal filtrada paso baja

Figura 3.1: Diferencia de mensajes recibidos del ordenador y del robot



(a) Señal sin procesar (b) Señal filtrada paso baja

Figura 3.2: Detalle de diferencia de mensajes recibidos del ordenador y del robot

lineal en las posiciones, velocidades y torques recibidos.

De esta manera, se obtienen vectores de la misma longitud (en el tiempo) tanto enviados como recibidos.

Ráfagas También se percibió la llegada a ráfagas en los mensajes provenientes del robot. Esto no supone un problema para preparar los datos, pero sí lo es a la hora de realizar un controlador realimentado que requiere la información que proporciona el robot sin desfases. El controlador propuesto en este trabajo no es realimentado, pero de serlo, una posible solución sería implementar el controlador en el propio robot (no ejecutarlo en el ordenador), aunque para ello habría que tomar nuevamente la base de datos o ajustarla a la frecuencia de generación de mensajes del robot. Otra posible solución consistiría en un predictor de posiciones y velocidades que ofrezcan al controlador esta información cuando no disponga de la misma. Por ejemplo, supongamos que en un instante concreto, llega una ráfaga de tres posiciones, velocidades y torques provenientes del robot. Esto significa que, dos instantes anteriores, el ordenador no ha obtenido ninguna información, por lo que la última posición, velocidad y torques conocidos son los obtenidos hace dos instantes temporales. En este caso, el predictor generaría 2 posiciones y velocidades (los torques serían la salida del sistema) para los instantes temporales que no disponen de información directa.

3.2.2. Red Neuronal

A continuación se estudia la viabilidad de la implementación de un controlador a partir del propio de Baxter basado en redes neuronales de aprendizaje profundo.

Viabilidad

El primer problema a enfrentarse es el de demostrar que la red neuronal es capaz de aprender algo de los datos extraídos del controlador del Baxter. Por la capacidad de detectar colisiones sabemos que es un sistema realimentado, posiblemente con un controlador PID.

Se podría realizar un controlador donde el modelo dinámico (controlador anticipativo) fuera desempeñado por la red neuronal, y donde la realimentación fuera desempeñada por el controlador PID. También podría implementarse la realimentación como otra red neuronal, pero ambas aproximaciones tienen el problema de enfrentarse a la realimentación y al tiempo real (a una frecuencia de realimentación de 100 Hz). Las redes neuronales profundas tienen el problema de ser pesadas en el cómputo (debido a las no linealidades usadas para la activación de las neuronas), poniendo en riesgo la capacidad de afrontar el problema en tiempo real.

Es por esto por lo que se decide hacer un controlador sin realimentación, donde los datos de entrada sean posiciones objetivo, posiciones en el momento de iniciar el movimiento, y velocidad objetivo, siendo los de salida una sucesión de torques a aplicar para alcanzar la posición deseada.

Para ello se elige una arquitectura basada en redes neuronales realimentadas, ya que las no realimentadas tienen la limitación de generar la misma salida ante la misma entrada, y siendo nuestra entrada única (debido a que el sistema no es realimentado), generaríamos siempre la misma salida, en vez de la sucesión de torques deseada.

Nos encontramos con el problema de si un modelo no realimentado, puede aprender del implementado por Baxter, que sí lo es. El controlador del Baxter consiste en un controlador mixto, formado por uno realimentado y otro anticipativo.

El controlador realimentado no es más que un sistema (lineal o no) que a ciertas entradas genera ciertas salidas (en función de su historia o no). El controlador anticipativo igualmente es un sistema que tiene en cuenta el modelo dinámico del robot, y por lo tanto los dos sistemas son reproducibles por una red neuronal. Por último, tenemos la realimentación, y es en este aspecto, donde la red neuronal tendrá que aprender a producir estimaciones de la salida provocada (posiciones de las articulaciones) al aplicar los torques.

Por lo tanto, la viabilidad de nuestro controlador depende de la capacidad de la red para generar dicha estimación de la realimentación.

Es por ello por lo que se utilizarán redes neuronales recurrentes, ya que tanto el controlador realimentado, como el anticipativo, como la realimentación en sí misma son sistemas que dependen de la historia de las señales de las que dependen, y esto es exactamente lo que hacen este tipo de redes.

Diseño

El diseño de la red se ha realizado haciendo uso de la librería *Keras*, que ofrece una capa de abstracción de diseño de redes profundas sobre *Tensorflow*.

El diseño consiste en las siguientes capas, una a continuación de la siguiente:

1. Capa de entrada. Esta capa aceptará matrices de nx15, donde n es la cantidad de datos a procesar en cada grupo de entrenamiento, y 15 es el número de dimensiones de entrada (7 de las posiciones deseadas, 7 de las posiciones de inicio, y 1 de la ratio de velocidad deseada).

2. Capa de repetición de vector. La entrada consiste en un único conjunto de datos. Sin embargo, se desea obtener una salida con tantos instantes temporales como longitud tenga la secuencia de mayor tamaño, por lo que repetimos el vector de entrada en el tiempo.

3. Capa totalmente conectada. La entrada se enlaza con una capa de 64 neuronas con activación relu.

4. Dropout. Se realiza una selección de neuronas aleatoriamente haciendo uso de la técnica de dropout, para así regularizar la salida.

5. GRU. El número de capas, así como el número de neuronas variará en función del experimento que se esté realizando. La activación es relu, y cuenta con mecanismos de dropout entre las conexiones recurrentes, así como regularización L1 y L2.

6. Dropout.

7. (Capa de convolución 1D + Dropout). Esta capa se activará o desactivará en función del experimento que se realice.

8. 2 capas totalmente interconectadas. De 500 neuronas cada una y activación relu

9. Dropout.

10. Capa de salida (totalmente interconectada). De tamaño 22 (7 de torque, 7 de posición, 7 de velocidad, y 1 de la máscara).

Se ha elegido GRU en vez de las celdas LSTM porque requiere de un menor cómputo, y permite hacer más iteraciones en la misma cantidad de tiempo, ofreciendo resultados similares.

La capa de convolucion 1D es un tipo de capa que opera sobre subconjuntos agrupados de entradas. Sirven para extraer dependencias espaciales en el conjunto de datos.

En la capa de salida, se han añadido las posiciones y velocidades obtenidas en cada momento, a fin de que el error generado por las mismas ayude a la red a aprender. Esto se ha realizado porque existe una dependencia entre los torques generados y la velocidad y posición que el brazo obtiene en cada momento. Los valores que realmente nos interesan son los de torque y máscara, por lo que se ha ponderado la aportación de las posiciones y velocidades por 0.1.

La capa de máscara sirve para delimitar la cantidad de torques válidos en cada secuencia. Esta capa es necesaria, ya que cada movimiento es de una longitud temporal distinta. Una vez entrenada la red generaremos torques de longitud variable, y sabremos qué cantidad de esos torque utilizar gracias a esta máscara.

En la fase de compilación del modelo, se ha elegido el optimizador Adam, y el tipo de error empleado ha sido el error absoluto promedio para los torques, y la entropía cruzada para la máscara (que representa la probabilidad de que el torque actual forme parte del movimiento necesario para alcanzar la posición deseada).

Experimentos

Una parte importante en el aprendizaje automático es la modificación y elección de parámetros (como la cantidad de capas, de neuronas por capa, la aportación al error de los métodos de regularización L1 y L2, así como la probabilidad de desactivar una neurona por el método dropout).

Para ello, se dividen los datos en 3:

- Conjunto de entrenamiento. Utilizado para entrenar la red.
- Conjunto de validación. Para obtener una medida del error de la red para cada uno de los parámetro a modificar.
- Conjunto de test. Para obtener una medida independiente de la elección del modelo y los parámetros que indique la eficacia de la red.

El problema de dividir en conjunto de entrenamiento y de validación, es que el error obtenido en el conjunto de validación depende del propio conjunto, y por lo tanto no es una medida totalmente fiable para seleccionar los parámetros. Es por esto por lo que se utiliza la técnica de K-pliegues (K-folding). Esta técnica consiste en realizar la división de un único conjunto (además del conjunto de test) en k subconjuntos. Después se realizará el entrenamiento de la red con todos menos con el un conjunto, y se medirá el error de la red sobre este conjunto. El entrenamiento se realiza excluyendo cada vez un subconjunto diferente (y añadiendo el anterior al conjunto de entrenamiento). Al final del proceso, obtendremos un erro medio, así como una desviación típica de la misma.

En base a los datos obtenidos con este método, se elegirá la red que mejores resultados ofrezca, y se medirá su error sobre el conjunto de test.

Los parámetros probados son:

- GRU.
  - 2 capas de 100 neuronas cada una.
  - 1 capa de 1000 neuronas.
- Regularización L1/L2.
  - L1 ó L2 con nivel de regularización 1e-6.
  - L1 ó L2 con nivel de regularización 0.001.
- Dropout.
  - Probabilidad de anular la neurona: 0.3.
  - Probabilidad de anular la neurona: 0.5.
  - Probabilidad de anular la neurona: 0.6.
  - Probabilidad de anular la neurona: 0.8.
- Convolución 1D.
  - Activada.
  - Desactivada.

Preparación de los datos

Para entrenar la red, hace falta preparar los datos. Esta preparación consiste en dividirlos en movimientos, normalizarlos (para un entrenamiento más rápido y robusto de la red), e igualar la longitud de las secuencias, ya que Keras necesita que todos los subgrupos que se utilicen para entrenar la red tengan la misma longitud.

También se creará la máscara, que valdrá 1 para todos los valores de torque que no se hayan puesto a 0, y 0 para los valores de relleno. Se utilizará esta máscara para el cálculo del error, ya que de no hacerlo, la red tendría en cuenta los errores cometidos en la sección de relleno.

3.3. Resultados

A continuación se muestran los resultados obtenidos de la anterior experimentación.

3.3.1. Base de datos

Por motivos de tiempo de entrenamiento, así como para aislar potenciales problemas a la hora de realizar el entrenamiento, se han tomado distintas bases de datos.

Todas las articulaciones
: El grueso de nuestra base de datos se ha realizado a partir del movimiento de todas las articulaciones, a distintas velocidades, con un tiempo de espera de 15 segundos.

Una articulación
: Se ha aislado el problema del tamaño del espacio a muestrear (las siete articulaciones, cada una con su rango de movimiento) reduciéndolo a una de las siete articulaciones en todo su rango de movimiento. El resto de las articulaciones se mantienen quietas en la posición cero.

5 segundos
: En lugar de limitar el tiempo para cada movimiento a 15 segundos, se limita a 5 segundos, para así aislar el problema de desdoblar la red en un intervalo de tiempo tan alto ($15\,\mathrm{s} * 100\,\mathrm{Hz} = 1500\,\mathrm{muestras}$).

0.5 seg. ratio vel. 1
: Es el caso más simple para entrenar la red. Se mueve una sola articulación (el resto se ubican en la posición 0) a máxima velocidad (relación de velocidad 1) durante 0.5 segundos. De esta manera obtenemos una base de datos muy amplia (más de 120 posiciones objetivo por minuto) con el problema del tamaño del espacio y del desdoblamiento temporal muy limitados.

Adicionalmente, se ha obtenido un conjunto de datos con combinaciones de distintas articulaciones, así como del brazo con las pinzas anexas para un tiempo máximo de 15 segundos y velocidad variable.

El tamaño total de la base de datos sin procesar es de 5 GB. A una velocidad de generación media de 94 KB/s, es un total de quince horas y media de recogida de datos.

3.3.2. Red neuronal

Se realizaron los experimentos descritos sobre la red neuronal. Por motivos de tiempo de entrenamiento, así como de obtención de la base de datos, se ha realizado el experimento con el conjunto de datos de 0.5 segundos por movimiento (unas 50 muestras por movimiento) y ratio de velocidad 1, esto es, a velocidad máxima.

Los resultados de todos los experimentos son los que se muestran en las figuras 3.3, 3.4 y 3.5. Estas gráficas se han obtenido haciendo uso del programa *tensorboard*.

En el eje horizontal se muestra el tiempo relativo durante el entrenamiento, mientras que en el eje vertical se observa el error. Observamos el error

cometido durante la fase de entrenamiento de las redes neuronales descritas con anterioridad.

Por motivos de espacio, no se muestran todas las gráficas para cada una de las configuraciones. En su lugar, se muestran las gráficas con los resultados de todas las redes y se explican los comportamientos de cada una de las redes, y por qué se comportan como se comportan en el entrenamiento.

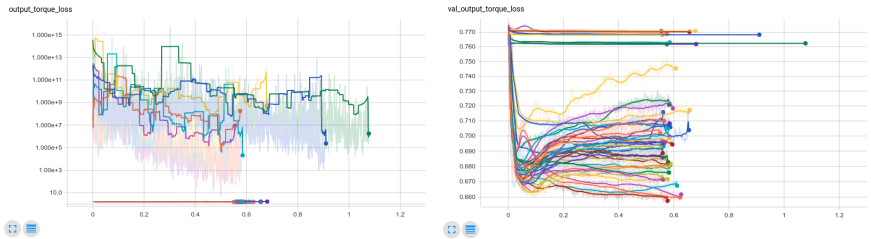
Error en el torque

Como se observa en la figura 3.3a, el error de entrenamiento es muy grande para algunas configuraciones de la red. Estas configuraciones corresponden con los modelos con un nivel de dropout de 0.8. Este nivel tan alto provoca en el entrenamiento que la salida no tenga nada que ver con la salida que generaría la red de no anularse las neuronas, y ésto es lo que hace que el error de entrenamiento sea tan alto.

Es la misma configuración la que en el conjunto de validación (fig. 3.3b) devuelve valores constante en torno a 0.76. Esto significa que la red está siendo infrautilizada, y no está aprendiendo. Lo mismo ocurre para la configuración de dropout igual a 0.6 y los valores de regularización L1 de 0.001 y de 1e-6. Los motivos del nivel de dropout a 0.6 son los mismos que para el nivel 0.8, mientras que el motivo por el cual la regularización L1 no funciona como se espera es que, en este caso, el tener neuronas especializadas a costa de apagar otras no funciona.

El resto de redes se comportan de manera similar en cuanto a torque se refiere. Las que mejor resultado dan son las que usan un nivel de regularización L2 de 0.001, 1000 neuronas y 1 capa, y un nivel de dropout de 0.5. Las redes de 2 capas y 100 neuronas no funcionan tan bien como las de 1 y 1000, debido a que las conexiones a través del tiempo de las celdas GRU está generando un fenómeno de sobre-aprendizaje sobre la base de datos de entrenamiento. Esto significa que está aprendiendo a reproducir los patrones de ruido temporales de las señales de entrenamiento.

Por otro lado, el nivel de regularización L2 de 1e-6 parece insuficiente para evitar este sobre-aprendizaje de la red. El sobre-aprendizaje se observa



(a) Entrenamiento (b) Validación

Figura 3.3: Errores en el torque

en que la curva de aprendizaje para el conjunto de entrenamiento sigue bajando, mientras que en el conjunto de validación, sube, lo que significa que sigue aprendiendo de la base de entrenamiento, pero lo aprendido no mejora (e incluso empeora) los resultados en datos no vistos antes.

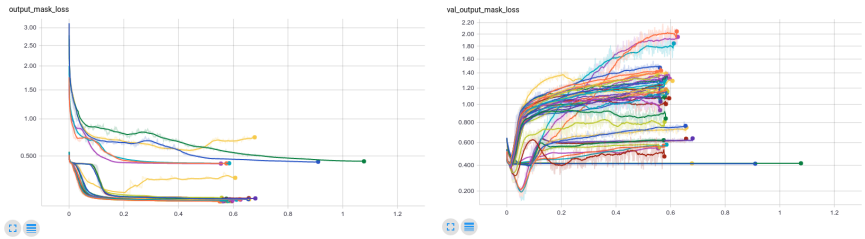
También se observa que, aunque está aprendiendo, el error es muy grande (en torno a 0.7). Hay que recordar que la salida está normalizada, por lo que tiene media 0 y desviación típica 1. Por lo que un error de estas dimensiones significaría que no está aprendiendo casi nada.

En realidad, lo que está ocurriendo es que la mayoría de articulaciones (todas menos una) se mantienen quietas, por lo que el rango de torques que están aplicando es mínimos. Esto provoca que, al normalizarlos, se amplifique en gran escala el ruido de los sensores al leer el torque (que tendría que ser para algunas articulaciones prácticamente constante). El problema es que el error generado por ese ruido afecta al error total igual que las articulaciones que sí tienen un rango más grande de operación y cuya relación señal ruido es mucho menor.

Este problema, una vez conocido, no debe preocuparnos, ya que las redes buscan patrones, y el ruido es la ausencia de dichos patrones, por lo que el aprendizaje no se verá afectado de manera tan negativa como parece en las gráficas.

Error en la máscara

Este fenómeno de sobre-aprendizaje es especialmente notable en la salida de la máscara (fig. 3.4). En ella se observa cómo, en el conjunto de entrenamiento, el error baja hasta niveles cercanos a 0, mientras que en la validación estos errores crecen por encima de 0.5. Estos errores tan altos en la máscara se deben a que realmente no hay una relación entre la entrada y la máscara en la base de datos elegidas (1 articulación moviéndose a máxima velocidad durante 0.5 segundos), sino que la máscara depende de la precisión con la que se generaban los mensajes en el Baxter, ya que los movimientos nunca llegan a la posición destino con velocidad 0 (condición para empezar el siguiente movimiento), sino que agotan todo el tiempo en intentar llegar a



(a) Entrenamiento (b) Validación

Figura 3.4: Errores en la máscara

la posición objetivo. Esto hace que la máscara tome valores de 50 y 51 para la mayoría de muestras, siendo así una medida independiente de la señal de entrada, y por lo tanto no la pueda aprender la red.

Es por esto por lo que se decide no tener en cuenta las medidas del error que la máscara genera, y centrar toda la atención en los errores generados por el torque.

Error total

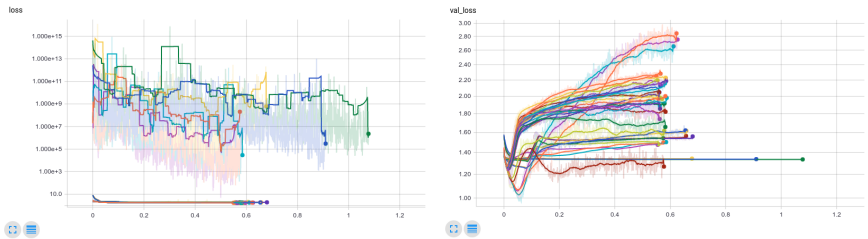
El error total (fig. 3.5) es la suma ponderada de los errores cometidos en los torques, la máscara, la posición y la velocidad. Los dos primeros suman al error multiplicados por 1, mientras que los otros dos multiplican por 0.1. Debido a que el objetivo de la red neuronal es obtener unos torques y una máscara adecuada, se han omitido las gráficas de las otras dos fuentes de errores.

Comparación de torques generados

En la figura 3.6 se muestran los errores de entrenamiento y validación para cada uno de los subconjuntos usando la técnica de k-fold. La red neuronal elegida es la dicha anteriormente: 1 capa y 1000 neuronas, valor de regularización L2 0.001, dropout 0.5, sin capa convolucional ni error L1.

Debido a que se detectó que los movimientos no acababan (la razón por la que la máscara no se podía aprender, y la mayoría de secuencias tenía una longitud de 50-51 muestras), se decide incluir la velocidad inicial del movimiento, ya que, si no termina el movimiento, significa que lleva una velocidad asociada, por lo que mejoraremos el rendimiento de la red al introducir esta información que antes suponíamos nula (velocidad 0 en todas las articulaciones).

Se observa que la red aprende adecuadamente a extraer los patrones de la base de datos (error decreciente con el número de iteraciones). Además, se observa que se ha conseguido disminuir el efecto del sobre-aprendizaje,

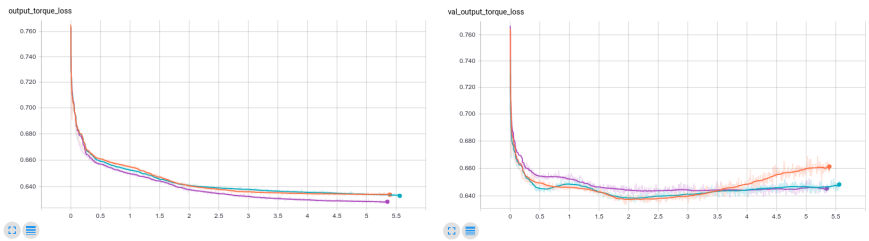


(a) Entrenamiento (b) Validación

Figura 3.5: Errores totales

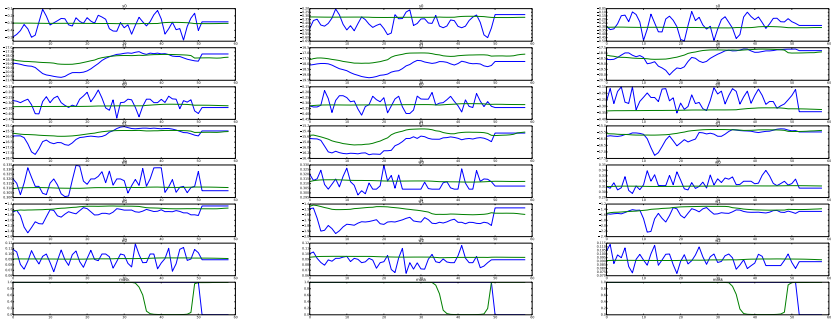
ya que el error de validación apenas crece una vez alcanzado el mínimo, y que el error de entrenamiento es comparable al de validación.

En la figura 3.7 se observan las predicciones que realiza la red. Se observa que la red es capaz de seguir la tendencia de baja frecuencia de cada una de las articulaciones. Se observa que, en articulaciones como la s0, donde el rango de operación es muy pequeño, la red ignora las perturbaciones (ruido) y sigue la tendencia general. En las articulaciones dondel el rango es mayor, esto también es cierto. Además, en estas últimas, se observa que las saltos grandes no los sigue. Estos cambios bruscos de torque se corresponden con la aportación que hace el controlador realimentado, y depende de una señal externa, y por lo tanto, incorrelada con el sistema que se pretende controlar. Es por esto por lo que la red no es capaz de aprender dichos cambios.



(a) Entrenamiento (b) Validación

Figura 3.6: Errores en el torque



(a) Entrenamiento (b) Validación (c) Test

Figura 3.7: Errores en el torque

Capítulo 4

Conclusiones

4.1. Trabajo realizado

4.2. Objetivos alcanzados

4.3. Trabajo futuro

Bibliografía

[1] Open Source Robotics Foundation. *Página principal de ROS*, (último acceso 6 de Abril 2016).

[2] Rethink Robotics. *Página principal de Baxter*, (último acceso 13 de Mayo 2016).

[3] Python Software Fundation. *Página principal de Python*, (último acceso 30 de Noviembre 2016).

[4] Google Brain Team. *Página principal de Tensorflow*, (último acceso 1 de Octubre 2016).

[5] Open ended Neuro-Electronic Intelligent Robot Operating System. *Página principal de Keras*, (último acceso 5 de Diciembre 2016).

[6] James Bowman. Tim Field, Jeremy Leibs. *Página principal de* `rosbag`, (último acceso 30 de Noviembre 2016).

[7] Gill A Pratt and Matthew M Williamson. Series elastic actuators. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 1, pages 399–406. IEEE, 1995.

[8] Coppelia Robotics. *Página principal de V-REP*, (último acceso 10 de Enero 2016).

[9] Open ended Neuro-Electronic Intelligent Robot Operating System. *Página principal de Gazebo*, (último acceso 24 de Febrero 2016).

[10] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook Singapore, 2012.

[11] Geoffrey Hinton. *Redes neuronales para aprendizaje automático*, (último acceso 8 de Octubre 2016).

[12] Andrew NG. *Aprendizaje automático*, (último acceso 20 de Septiembre de 2015).

[13] Sebastian Ruder. *Diferencias entre algoritmos de optimización*, (último acceso 10 de Noviembre 2016).

[14] Christopher Olah. *La eficacia irracional de las redes neuronales recurrentes*, (último acceso 17 de Septiembre 2016).

[15] Christopher Olah. *Entendiendo las redes LSTM*, (último acceso 17 de Septiembre 2016).

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[18] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[19] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[20] Yarin Gal. A theoretically grounded application of dropout in recurrent neural networks. *arXiv preprint arXiv:1512.05287*, 2015.

[21] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[22] Google. *Apredizaje profundo*, (último acceso 15 de Julio 2016).

[23] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

[24] BVLC. *Web principal de Caffe*, (último acceso 15 de Enero 2016).

[25] Koray Ronan, Clement and Soumith. *Web principal de Torch*, (último acceso 14 de Enero 2016).

[26] LISA lab. *Web principal de Lasagne*, (último acceso 16 de Enero 2016).

[27] Bohdan T Kulakowski, John F Gardner, and J Lowen Shearer. *Dynamic modeling and control of engineering systems*. Cambridge University Press, 2007.

[28] Coleman Brosilow and Babu Joseph. *Techniques of model-based control*. Prentice Hall Professional, 2002.

[29] Silvia Tolu, Mauricio Vanegas, Jesus A Garrido, Niceto R Luque, and Eduardo Ros. Adaptive and predictive control of a simulated robot arm. *International journal of neural systems*, 23(03):1350010, 2013.

[30] Silvia Tolu, Mauricio Vanegas, Niceto R Luque, Jesús A Garrido, and Eduardo Ros. Bio-inspired adaptive feedback error learning architecture for motor control. *Biological cybernetics*, 106(8-9):507–522, 2012.

[31] Daniel E Rivera, Manfred Morari, and Sigurd Skogestad. Internal model control: Pid controller design. *Industrial & engineering chemistry process design and development*, 25(1):252–265, 1986.

[32] Duarte Valério and José Sá da Costa. Tuning of fractional pid controllers with ziegler–nichols-type rules. *Signal Processing*, 86(10):2771–2784, 2006.

[33] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[34] Ilber A Ruge and Miguel A Alvis. Aplicación de los algoritmos genéticos para el diseño de un controlador pid adaptativo. *Revista Tecnura*, 13(25):81–87, 2012.