UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

Trabajo Fin de Grado

CURSO 2018/2019

APLICACIÓN DE TÉCNICAS DE DEEP LEARNING ON-LINE AL APRENDIZAJE DE MODELOS DE MUNDO DE ROBÓTICA COGNITIVA

Grado en Ingeniería Mecánica

ALUMNA/O

Carmen Boado de la Fuente

TUTORAS/ES

Francisco Javier Bellas Bouza

Alejandro Romero Montero

FECHA

SEPTIEMBRE 2019

1 TÍTULO Y RESUMEN

1.1 Título

Aplicación de técnicas de *Deep Learning on-line* al aprendizaje de modelos de mundo de robótica cognitiva.

1.2 Resumen

Este trabajo fin de grado se enmarca en el proyecto europeo de investigación DREAM llevado cabo en el Grupo Integrado de Ingeniería (GII) de la UDC, y centrado en el uso de modelos cognitivos humanos aplicados al desarrollo de capacidades avanzadas en robots autónomos. Uno de los elementos desarrollados en este proyecto fue el sistema de aprendizaje de modelos de mundo dentro de la arquitectura cognitiva llamada *Multilevel Darwinist Brain* (MDB), imprescindible para poder realizar procesos de razonamiento complejos. Hasta el momento, en el marco del DREAM, se ha abordado el aprendizaje de estos modelos utilizando algoritmos evolutivos, pero son computacionalmente costosos para su uso en tiempo real.

En este trabajo de fin de grado se plantea el estudio de aplicación de técnicas de *Deep Learning on-line*, utilizando para ello la herramienta *Tensor Flow*. Se propondrá un nuevo procedimiento de aprendizaje *on-line* de modelos de mundo, con un nuevo método de gestión de memoria, y se analizarán los parámetros óptimos del modelo en un ejemplo típico de robótica cognitiva, midiendo el tiempo de cómputo y la precisión de red neuronal propuesta.

Título

Aplicación de técnicas de Deep Learning online á aprendizaxe de modelos de mundo de robótica cognitiva

Resumo

Este traballo de fin de grao enmárcase no proxecto europeo de investigación DREAM levado cabo no Grupo Integrado de Enxeñaría da UDC, e centrado no uso de modelos cognitivos humanos aplicados ao desenvolvemento de capacidades avanzadas en robots autónomos. Un dos elementos desenvolvidos neste proxecto foi o sistema de aprendizaxe de modelos de mundo dentro da arquitectura cognitiva chamada *Multilevel Darwinist Brain* (MDB), imprescindíbel para poder realizar procesos de razoamento complexos. Ata o momento, no marco do DREAM, abordouse a aprendizaxe destes modelos utilizando algoritmos evolutivos, pero son computacionalmente costosos para o seu uso en tempo real.

Neste traballo de fin de grao plantéxase o estudo de aplicación de técnicas de *Deep Learning on-line*, mediante a ferramenta Tensor Flow. Proporase un novo procedemiento de aprendizaxe *on-line* de modelos de mundo, cun novo método de xestión de memoria, y estableceranse os parámetros óptimos do modelo nun exemplo típico de robótica cognitiva, medindo o tempo de cómputo e a precisión da rede neuronal proposta.

Tittle

Application of online Deep Learning techniques to world model learning in cognitive robotics.

Summary

This final degree project is part of the European research project DREAM carried out in the Integrated Engineering Group (GII) of the UDC, and focused on the use of human cognitive models applied to the development of advanced capabilities in autonomous robots. One of the main elements of this project was the learning system of world models within the cognitive architecture called Multilevel Darwinist Brain (MDB), essential to perform complex reasoning processes. Until now, in the framework of DREAM, the learning of these models has been carried out using evolutionary algorithms, but they are computationally expensive for their use in real time.

In this final degree project, the application of on-line Deep Learning techniques is studied, using the Tensor Flow tool. A new procedure for on-line learning of world models will be developed, with a new method of memory management, and the optimal parameters of the model will be analyzed in a typical example of cognitive robotics, by measuring the computation time and the accuracy of the proposed neural network.

2 ÍNDICES

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

3 INTRODUCCIÓN

El Trabajo de Fin de Grado que se presenta a continuación se enmarca en el contexto de la robótica cognitiva y se centra en el proyecto DREAM, que, tal y como se relatará más adelante, abarcará el problema del aprendizaje en tiempo real por parte de un robot cognitivo.

La robótica cognitiva abarca la investigación y diseño de sistemas artificiales basados en modelos cognitivos humanos para lograr una respuesta inteligente. La adquisición de este comportamiento inteligente se realiza a través del aprendizaje autónomo de diferentes modelos que permiten que el robot se adapte a su entorno para lograr sus objetivos.

3.1 Cognitive Developmental Robotics

Tradicionalmente, el proceso de desarrollo de un robot ha estado predefinido totalmente por el diseñador. Por lo general, este se reducía al planteamiento de una tarea a resolver que era analizada por el propio diseñador. Este elegía un método computacional para su resolución, y finalmente lo implementaba en el robot. Los parámetros estudiados en el método computacional de resolución estaban ajustados para una tarea concreta, y no se podían generalizar para otras, lo cual suponía un problema a largo plazo. Con el paso de los años, este procedimiento se ha vuelto cada vez más complejo al tratar de resolver problemas más ambiciosos. Es aquí donde surge la aproximación de la Robótica Cognitiva del Desarrollo, en inglés, *Cognitive Developmental Robotics* (CDR) [1].

El campo de CDR se basa en procesos de aprendizaje abiertos (*open-ended*) que se llevan a cabo en tiempo real, y en los que se hace uso de diferentes mecanismos inspirados en modelos de desarrollo cognitivo humano. El objetivo final de CDR es que el robot sea realmente autónomo en todo su proceso de aprendizaje, descubra y seleccione diferentes objetivos por sí mismo, a la par que evalúe la exactitud del conocimiento y funcionamiento aprendidos. Existen una serie de propiedades básicas que caracterizan a las arquitecturas CDR [2]:

- Las características físicas del robot constituyen una parte de las restricciones en la interacción entre él mismo y el entorno. Este concepto se conoce como *Physical Embodiement*.
- El robot se debe ubicar en un entorno físico o social.
- Derivado de las interacciones con el entorno físico o social, y en un periodo prolongado, el sistema debe de ser capaz de crear y desarrollar estructuras cognitivas más complejas.
- Los procesos en los que se basa CDR se implementan en forma de arquitectura cognitiva, un sistema computacional que se inspira en la estructura biológica del cerebro, centrando la atención en las estructuras de las zonas en las que se desarrollan las conexiones e interacciones entre ellas. Una buena arquitectura CDR debería desarrollar modelos y no imitarlos.

Las arquitecturas cognitivas en CDR se fundamentan en el aprendizaje de modelos y habilidades (*skills*), que se deben llevar a cabo en tiempo real mientras el robot opera en el entorno. Dichos modelos son la base para los procesos de razonamiento interno que realiza el robot. El formalismo matemático más habitual en CDR se inspira en el campo del aprendizaje por refuerzo, que revisaremos a continuación.

3.2 Reinforcement Learning

Se trata de una rama perteneciente al aprendizaje automático que presenta un conjunto de métodos cuyo fin es descubrir cuáles son las acciones que llevan al sistema autónomo a obtener una recompensa máxima, sin que se le asigne explícitamente al agente ningún conocimiento previo [3][4].

Los modelos de aprendizaje por refuerzo se pueden describir como un agente en contacto con un medioambiente o entorno, recibiendo información a través de un estado y una recompensa, que repercutirá en la realización de una acción, bien positiva o bien negativa (refuerzo). En la siguiente figura se puede observar el ciclo de funcionamiento básico del Aprendizaje por Refuerzo:

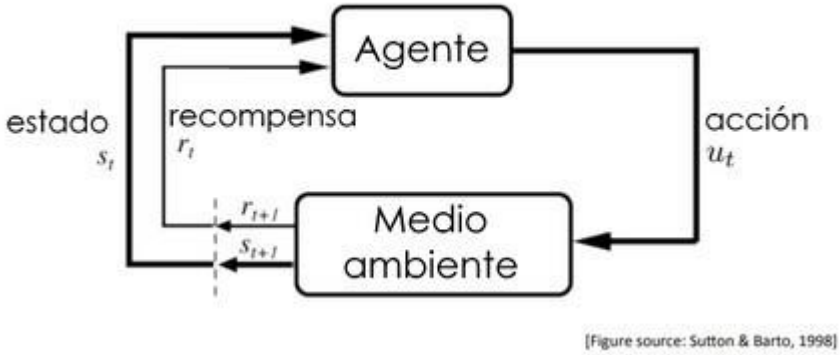

Figura 3-1. Esquema básico del funcionamiento de Aprendizaje por Refuerzo

Una vez conocidos el agente y el entorno o medio ambiente, se puede encontrar un total de cuatro elementos principales derivados del proceso de aprendizaje por refuerzo:

- Una **política** describe cómo se debería comportar el agente para realizar las acciones que le lleven a conseguir la máxima recompensa.
- La s**eñal de recompensa** es el elemento que va a definir el objetivo del problema a corto plazo. Esta señal numérica, en función de su valor, marcará qué acciones son buenas o malas para el agente.
- La **función de valor**, a diferencia de la señal de recompensa, especifica lo que es bueno, pero a largo plazo. Aunque un estado produzca una recompensa baja, el valor puede seguir siendo alto.
- El **modelo** del entorno simula el comportamiento del entorno, permitiendo hacer deducciones de cómo se comportará el mismo entorno.

El este campo, el objetivo final es el aprendizaje de las políticas, que definen las acciones a realizar por el robot a partir del estado sensorial en el que se encuentra. Para el aprendizaje de estas políticas existen diversas aproximaciones, aunque se pueden agrupar en dos grandes tipos: basadas en modelos o libres de modelos.

Los métodos basados en modelos (*model-based methods*) agrupan aquellos algoritmos que realizan un número mínimo de interacciones con el entorno real para después construir un modelo basado en dichas interacciones. La ventaja de construir un modelo es que permite un aprendizaje más rápido, ya que no es necesario esperar una respuesta por parte del entorno. Al contrario que los métodos basados en modelos, los métodos sin modelos (*model-free*) llevan a cabo el aprendizaje en base a las experiencias con el mundo real.

En el caso de CDR, por lo comentado en el apartado anterior, solo tiene sentido utilizar una aproximación basada en modelos, ya que la adquisición del conocimiento se realiza de manera abierta y progresiva. Pero, además, CDR añade la necesidad de que dicho

aprendizaje se lleve a cabo en tiempo real, algo que no es muy común en aprendizaje por refuerzo.

Para afrontar este problema del aprendizaje por refuerzo basado en modelos y en tiempo real dentro de una arquitectura cognitiva, diferentes autores han comenzado a aplicar técnicas de aprendizaje profundo (*Deep Learning*) [5]. En un apartado posterior se comentará con más detalle este tipo de aproximación, pero lo importante es que se están obteniendo resultados muy satisfactorios. Los investigadores del GII, dentro del proyecto europeo DREAM [6]., han venido desarrollando una arquitectura cognitiva, denominada *Multilevel Darwinist Brain* (MDB) [7]., en la que el problema del aprendizaje de modelos en tiempo real no está, ni mucho menos, resuelto. Hasta el momento se estaban utilizando algoritmos evolutivos para este proceso, pero su excesivo tiempo de cómputo los hace poco aplicables a problemas complejos. De aquí surge, pues, la motivación de este TFG, en el que se estudiarán las técnicas basadas en *Deep Learning on-line* para ser aplicadas en el MDB.

4 OBJETIVOS

El objetivo principal de este proyecto es el estudio de la aplicabilidad de técnicas de *Deep Learning* on-line para el aprendizaje de modelos dentro de la arquitectura MDB que se desarrolla en el proyecto europeo DREAM.

4.1 Subobjetivos

Para lograr este objetivo global, se han planteado los siguientes subobjetivos:

- Analizar el proceso de aprendizaje actual en la arquitectura MDB.
- Analizar los algoritmos de aprendizaje *on-line* para redes neuronales dentro del campo de *Deep Learning*.
- Desarrollar un procedimiento de aprendizaje on-line teniendo en cuenta las limitaciones en el tiempo de cómputo.
- Realizar un estudio comparativo formal de las diferentes opciones en ejemplos de aplicación típicos.
- Proponer un procedimiento de gestión de la memoria a corto plazo y una parametrización del algoritmo de aprendizaje seleccionado.

5 FUNDAMENTOS TEÓRICOS Y FUNDAMENTOS TECNOLÓGICOS

A continuación, explicaremos los aspectos teóricos que ha sido necesario estudiar para el desarrollo de este Trabajo de Fin de Grado. Se comenzará explicando qué es el *Deep Learning* y las diferentes versiones del algoritmo de descenso de gradiente que se usan en este campo y se continuará con la herramienta utilizada para el desarrollo computacional del trabajo, *TensorFlow*, y su API de nivel superior *Keras*.

5.1 Deep Learning

Aproximadamente en el año 2006, el *Deep Learning* (Aprendizaje Profundo) surgió como un campo independiente perteneciente al *Machine Learning* (Aprendizaje Automático), que a su vez se engloba dentro de un campo más amplio que es el de la Inteligencia Artificial (Figura 5.1). Aún hoy en día no existe una definición general [5] y extendida para describir el *Deep Learning*, pero podríamos decir que es una generalización de los sistemas de aprendizaje basados en redes neuronales en la que se utilizan arquitecturas con varias capas, que son capaces de procesar datos y crear patrones que se pueden utilizar en tomas de decisiones de más alto nivel. Por decirlo de una forma simple, una red neuronal profunda es una caja negra con entradas y salidas, cuyos parámetros se ajustan mediante un algoritmo de aprendizaje profundo, y que es capaz de aprender relaciones muy complejas entre dichas entradas y salidas.

El auge del *Deep Learning* se debe a dos factores principales: la existencia de grandes conjuntos de datos etiquetados, sobre todo imágenes, y el aumento en la capacidad de cómputo de los procesadores, mayormente las *GPUs*.

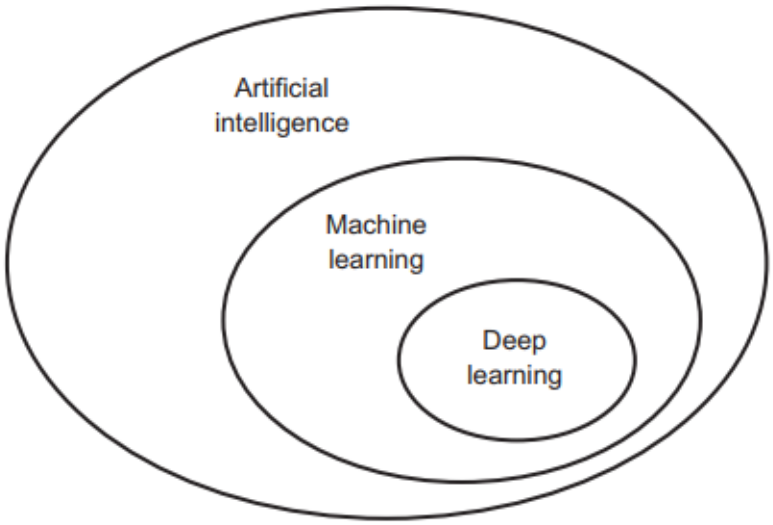

Figura 5-1. Esquema de clasificación de la IA.

Dentro del *Deep Learning* existen numerosos algoritmos de optimización de los parámetros de las redes neuronales, y la mayoría basan su funcionamiento en encontrar los parámetros del modelo que minimizan una función de pérdida con unos datos de

entrenamiento. El algoritmo más extendido se denomina Algoritmo de Descenso de Gradiente[8].

5.1.1 Algoritmo de descenso de gradiente

El algoritmo de descenso de gradiente es un algoritmo clave dentro del ámbito de *Machine Learning*. Se puede definir algoritmo de descenso de gradiente como un algoritmo de optimización iterativo cuya función es hallar el mínimo de una función convexa y diferenciable $F(\theta)$ para un dominio determinado R^d [8].

El funcionamiento de algoritmo consiste en encontrar un mínimo a partir de un punto P dentro del dominio de la función. Partiendo de este punto P, es decir, de los valores de los parámetros aleatorios que definen la función, y tomando pasos de un tamaño determinado en sentido contrario al gradiente, se intenta llegar a este mínimo. Estos pasos tienen su tamaño determinado por la tasa de aprendizaje η. Hasta que la función se estabilice (hasta que se llegue a un valle de la función) el algoritmo seguirá iterando, de tal forma que se sigue la dirección de la pendiente de la superficie creada por la función $F(\theta)$, que será la función de costo, o función de pérdida, hasta el punto de estabilización comentado anteriormente.

Para encontrar el mínimo se recurre a la derivada de la función, que nos indica la pendiente de la función en ese punto. La actuación de este algoritmo se puede derivar del algoritmo básico del aprendizaje online, descrito como:

$$\theta = \theta - \nabla_\theta F(\theta)$$

Sin embargo, el descenso de gradiente tiene en cuenta la tasa de aprendizaje, que es un concepto clave en el desarrollo del algoritmo. Una de las cuestiones más importantes es indicar el valor que deberá tomar la tasa de aprendizaje, es decir, de qué tamaño se tiene que ir dando los pasos hasta llegar al mínimo. Una alta tasa de aprendizaje influirá en la velocidad y tiempo necesarios hasta que se llegue al mínimo, de tal manera que el tiempo será mucho mayor si la tasa de aprendizaje tiene un valor muy bajo. Sin embargo, un valor alto puede desembocar en el problema de la no convergencia. Por otro lado, un valor muy bajo puede llevar al algoritmo a encontrar y estancarse en un mínimo local y no global.

Derivadas del algoritmo de descenso de gradiente original, aparecen las siguientes ramificaciones:

- Descenso de gradiente por lotes.
- Descenso de gradiente estocástico
- Descenso de gradiente por *mini-batches* o mini-lotes

5.1.1.1 Descenso de gradiente por lotes

El descenso de gradiente por lotes (*Batch Gradient Descent*) calcula el gradiente de la función de coste con respecto a los parámetros que definen la función (θ). Siendo:

- θ el conjunto de parámetros que definen la función
- $F(\theta)$ la función de coste que define un problema
- η la tasa de aprendizaje

podemos definir el funcionamiento del algoritmo de descenso de gradiente por lotes de manera matemática como:

$$\theta = \theta - \eta * \nabla_\theta F(\theta)$$

Un problema que se puede observar derivado del uso del algoritmo de descenso de gradiente por lotes es que no permite el aprendizaje online debido a que la actualización del gradiente se produce sobre el conjunto de datos completo. Esto implica unos bajos rendimiento y velocidad en el caso de ser utilizado en tiempo real.

En este TFG, lo que se pretende estudiar son técnicas para el aprendizaje online, en el cual la llegada de datos es continua y sobre la marcha, con lo cual se eliminó rápidamente la idea de trabajar con este algoritmo para la realización de pruebas.

5.1.1.2 Descenso de gradiente estocástico

En un amplio conjunto de datos aleatorios es posible la aparición de datos redundantes. De hecho, cuanto mayor sea el conjunto, la probabilidad de encontrar datos repetidos tiende a aumentar. El uso del gradiente de descenso implicaría utilizar el mismo dato en varias ocasiones.

Con el descenso de gradiente estocástico, abreviado como SGD, el cálculo de gradiente se realiza para un lote de tamaño unitario. En este caso se produciría una actualización para cada dato, aumentando la velocidad de cómputo y una convergencia más temprana y posibilitando el aprendizaje online.

Siendo *x* un ejemplo de entrenamiento con su etiqueta *y* correspondiente, podemos describir el algoritmo de manera matemática de la siguiente forma:

$$\theta = \theta - \eta * \nabla_\theta F(\theta; x^i; y^i)$$

Debido a que las actualizaciones se producen con cada dato, la gráfica de la pérdida contiene muchos más picos y fluctuaciones (se introducen muchos más ruidos en los pesos). Pero esta última característica no es necesariamente negativa, ya que permite al algoritmo salir de mínimos locales y evitar una convergencia prematura. Aparte, las frecuentes actualizaciones de los parámetros dan una idea del rendimiento del modelo.

Si bien hasta ahora se han presentado unas ventajas muy interesantes, se puede decir que la gran desventaja es que las constantes actualizaciones del modelo resultan computacionalmente mucho más costosas.

5.1.1.3 Descenso de gradiente por mini-lotes o mini-*batches*

Finalmente, la opción de uso más extendida entre estos tres algoritmos es el Descenso de Gradiente por mini-lotes. En este caso se separa el conjunto de entrenamiento en mini-lotes con *n* ejemplos de entrenamiento y realiza una actualización del gradiente para cada uno de ellos. Se puede describir el algoritmo como:

$$\theta = \theta - \eta * \nabla_\theta F(\theta; x^{i:i+n}; y^{i:i+n})$$

Esta opción resulta muy atractiva por varias razones:

- Las actualizaciones son más frecuentes que en el descenso de gradiente por lotes, lo que permite evitar mínimos locales y alcanzar una convergencia más robusta.
- Aunque computacionalmente sea más costoso que el descenso de gradiente por lotes, el descenso de gradiente utilizando mini-lotes es bastante más eficiente que un descenso de gradiente estocástico.
- El procesamiento de mini-lotes permite una gestión más eficiente de la memoria, ya que no es necesario mantener la totalidad del conjunto de datos almacenados.

En este TFG se realizará un estudio preliminar de cuál de estas técnicas es la más adecuada para el tipo de aprendizaje a realizar en la arquitectura MDB, combinando técnicas estocásticas con mini-lotes.

Aunque se hablará en el desarrollo más acerca de todo el proceso de aprendizaje, existen numerosos algoritmos que se basan en el funcionamiento de las tres opciones presentadas. Uno de ellos es el *Adam Optimizer* [9], en cual ha sido el algoritmo de aprendizaje para llevar a cabo las pruebas y experimentos propuestos en este trabajo.

5.2 TensorFlow

Tensorflow [10][11] es una biblioteca de código abierto para cálculos de flujos de datos construida por Google. Se trata de una interfaz que nos permite expresar e implementar algoritmos válidos para el aprendizaje automático. Esta interfaz se puede implementar en sistemas muy diferentes, como pueden ser dispositivos móviles, tabletas, robots o diferentes sistemas computacionales como las tarjetas GPU. Esta biblioteca de código abierto es enormemente flexible y sirve para expresar una amplia variedad de algoritmos, como pueden ser algoritmos de entrenamiento de redes, realización de investigaciones, implementación de sistemas de aprendizaje automático... La API de *TensorFlow* se lanzó como un paquete de código abierto sobre noviembre de 2015.

Figura 5-2. Logo TensorFlow

En 2011 Google Brain desarrolló *DistBelief*, como un sistema de aprendizaje automático basado en redes de aprendizaje profundo, que más tarde se simplificó para dar como resultado *TensorFlow*, más rápida y más flexible que su predecesora.

TensorFlow toma datos descritos utilizando un modelo similar al flujo de datos que se asignan a diversas plataformas de hardware. Puede ser ejecutado tanto en una sola máquina como en un conjunto de plataformas, tanto en la nube como en sistemas locales, lo que demuestra la gran escalabilidad que posee. El tener una herramienta que pueda abarcar un arco tan grande de plataformas simplifica enormemente el aprendizaje automático.

Esta biblioteca es lo suficientemente flexible como para poder experimentar de manera muy dinámica con diversos modelos en los campos de investigación del aprendizaje automático. Así permite la ejecución paralela de un modelo central de flujo de datos con numerosos dispositivos computacionales a su alrededor interactuando para actualizar estados compartidos, aparte de permitir diferentes enfoques con pocos cambios.

Debido a sus sobresalientes características, la librería de TensorFlow es la plataforma más utilizada en el campo del *Deep Learning*. Para el desarrollo de redes neuronales, se utilizó la API de alto nivel de *TensorFlow* denominada *Keras*.

5.2.1 Keras

Se trata de una API de alto nivel para desarrollar y entrenar modelos de *Deep Learning* escrita en Python y capaz de ejecutarse sobre *TensorFlow* (https://www.tensorflow.org/guide/keras). Su uso es muy variado: creación rápida de prototipos, investigación avanzada y producción. Se puede describir a partir de tres enfoques clave:

- Facilidad de uso: la simplicidad y la optimización para casos comunes representan una facilidad de uso para los usuarios.
- Modulable: los modelos de Keras están constituidos por bloques configurables que se pueden conectar con muy pocas restricciones.
- Fácil de extender: aparte de todas las posibilidades que ofrecen los modelos ya creados en Keras, también existe la posibilidad de crear nuevos módulos, lo que permite una gran adecuación en el campo de la investigación.

En este TFG se ha utilizado esta API para desarrollar el modelo de entrenamiento necesario para el aprendizaje de un conjunto de datos recogidos a partir del robot Baxter. La utilización de esta API no ha sido aleatoria, sino que, dado el conocimiento somero por parte del alumno en el entorno de programación, la facilidad de uso con el lenguaje de Python abrevia en tiempo invertido en las tareas experimentales en el TFG.

La simplicidad de uso de Keras se puede demostrar con la construcción de un modelo básico. Aunque durante todo el proceso de creación de este TFG se ha ahondado en diferentes bibliotecas donde se almacenan funciones predefinidas, el modelo más simple es el modelo *sequential()*, al que se le pueden agregar diferentes capas ocultas de un cierto número de neuronas con el comando *.add()*. Tras ello, con el comando *.compile()* se configura el proceso de entrenamiento y el modelo se entrenará con los comandos *.fit()* o *.train_on_batch*, si se quieren pasar lotes de manera manual. La evaluación se realiza con el comando *.evaluate()*.

Aunque existen numerosos algoritmos dentro de Keras que se podrían aplicar en este TFG, uno de los más extendidos actualmente para el aprendizaje online de redes neuronales es el *Adam Optimizer*, del que hablaremos con más detalle a continuación.

5.2.1.1 Adam Optimizer

El algoritmo *Adam Optimizer* [9] es un algoritmo de optimización basado en los gradientes de primer orden. Su implementación es muy sencilla. De hecho, al utilizar la API de Keras de *TensorFlow*, el método ya está implementado y simplemente se tiene que importar de la librería. Aparte, su eficiencia computacional y los requisitos mínimos de memoria lo hacen una opción muy aceptable para el aprendizaje online, que implica grandes conjuntos de datos.

Una característica importante en este algoritmo de optimización es que calcula las tasas de aprendizaje adaptativas para los parámetros a partir de la estimación del primer (media) y segundo (varianza) momento del gradiente.

Se puede ver un esquema de funcionamiento a continuación:

Inicio

Se pide:

- α
- $\beta_1 \; y \; \beta_2$. /*Tasas de descenso exponencial para el momento estimado*/.
- $F(\theta)$. /*Es la función objetivo*/
- θ_0. /*Se trata del vector inicial de parámetros de la función objetivo*/

Se define:

- $m_0 = 0$
- $v_0 = 0$
- $t = 0$

mientras θ_t no converja:

- $t = t + 1$
- $g_t = \nabla_\theta F_t(\theta_{t-1})$
- $m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$ /*Estimación del primer momento*/
- $v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$ /*Estimación del segundo momento*/
- $\widehat{m_t} = m_t / (1 - \beta_1^t)$ /*Corrección de m_t*/
- $\widehat{v_t} = v_t / (1 - \beta_2^t)$ /*Corrección de v_t*/
- $\theta_t = \theta_{t-1} - \alpha * \frac{\widehat{m_t}}{(\sqrt{\widehat{v_t}} + \epsilon)}$ /*Actualización de parámetros*/

Final del bucle

Se devuelve la última actualización de los parámetros θ_t

Aunque existen numerosos algoritmos de optimización aplicables a este TFG como puede ser el *AdaGrad* o el *RMSProp*, los beneficios y ventajas comentados anteriormente con respecto a otros algoritmos de optimización hacen del *Adam Optimizer* la herramienta adecuada para el desarrollo de este TFG. A menudo lo clasifican como un algoritmo de descenso de gradiente estocástico, pero la realidad es que puede funcionar perfectamente tanto con entradas de datos estocásticas como con mini-lotes.

5.3 Robot BAXTER

Baxter (http://sdk.rethinkrobotics.com/wiki/Home) se puede definir como un robot antropomorfo humanoide que cuenta con varios componentes entre los que se incluyen los brazos articulados mostrados en la figura 5-4, los cuales poseen siete grados de libertad. El robot incorpora sensores de última generación como elementos de control de fuerza, posición y par de cada articulación, cámaras preparadas para aplicaciones de visión artificial y varios elementos de entrada y salida preparados para el usuario como pueden ser mandos, botones, una pantalla en la zona de la cabeza…

El principal elemento sensor de Baxter es la cámara que posee en la cabeza, mostrada en la figura, de la cual se extrae información sensorial del entorno del robot (figura 5-3). A mayores, la unidad que posee el GII, tiene una cámara 3D situada en el techo del laboratorio que proporciona una visión en planta del entorno.

Figura 5-3. Cámara en zona de la cabeza.

A nivel de actuadores el Baxter está dotado principalmente de brazos y pinzas en sus extremos.

5.3.1 Grippers

Los *grippers* (pinzas) están formados por dos elementos que simulan un agarre en forma de mordaza y que se mueven de manera simétrica. Son capaces de soportar y elevar cargas de hasta 5 libras, lo que equivale aproximadamente a 2,27 kilogramos.

Dependiendo de la colocación de las pinzas se pueden obtener diferentes modalidades de agarre, con un gran abanico de velocidades y niveles de fuerza. La distancia de apertura máxima de la pinza es de 44 milímetros.

La precisión del *gripper* son 5 milímetros, por lo tanto, el error máximo que se debe obtener en los entrenamientos del modelo finales, para que sean satisfactorios, debe de ser de 5 milímetros.

5.3.2 Brazos

Son la parte más voluminosa del robot Baxter. Los movimientos de las articulaciones de los brazos en las direcciones de los ángulos de rotación de cabeceo y alabeo (*pitch y roll*) se pueden desarrollar gracias a los actuadores *Series Elastic Actuators* (SEAs).

5.3.2.1 Especificaciones de diseño de los brazos

Tal y como se muestra en las figuras 5-4 y 5-5, el brazo del robot Baxter consta de siete partes principales, señaladas con las letras S, E y W, siendo:

- S la señalización de las partes correspondientes al hombro.
- E la señalización de las partes correspondientes al codo.
- W la señalización de las partes correspondientes a la muñeca.

Los nombres de las articulaciones del brazo se indican de la siguiente manera:

S0 - *Shoulder Roll*

S1 - *Shoulder Pitch*

E0 - *Elbow Roll*

E1 - *Elbow Pitch*

W0 - *Wrist Roll*

W1 - *Wrist Pitch*

W2 - *Wrist Roll*

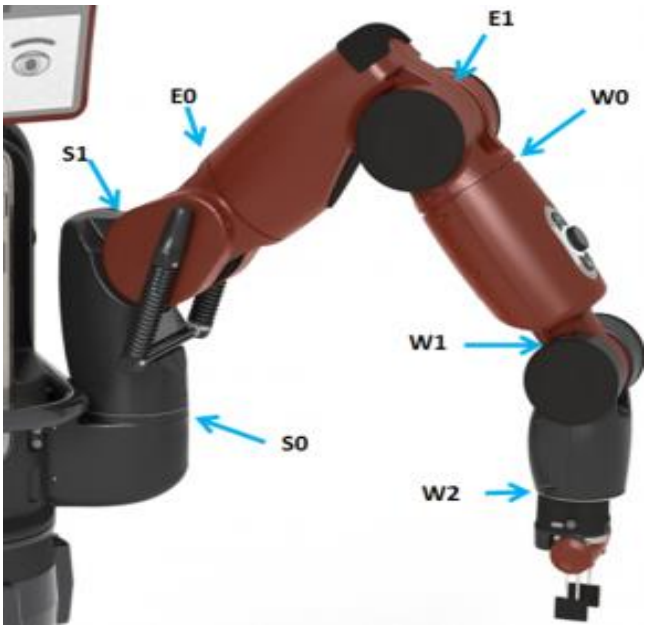

Figura 5-4. Partes del brazo Baxter.

El brazo del robot Baxter tiene siete grados de libertad en forma de ángulos, correspondientes a las siete entradas del archivo de conjunto de datos proporcionado por los investigadores del GII. Los rangos de movimiento para cada uno son limitados:

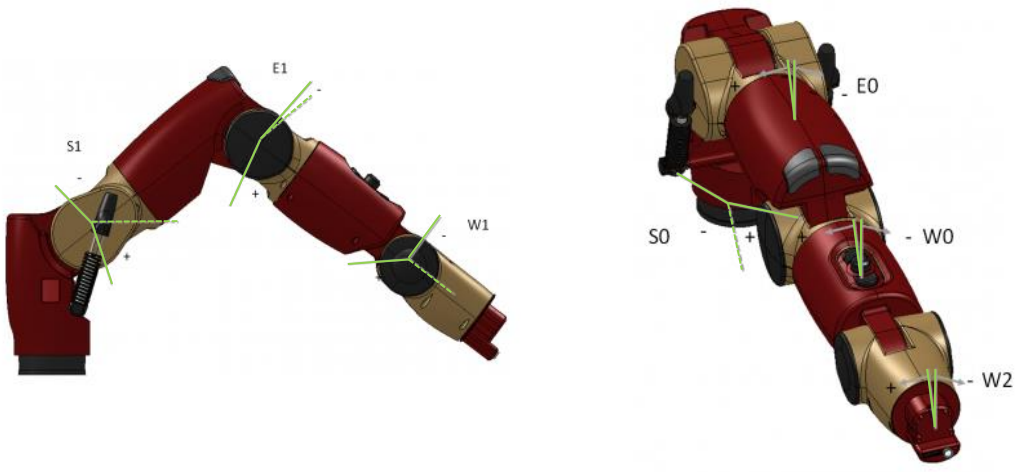

Figura 5-5. Grados de libertad del Robot.

5.3.2.2 Series Elastic Actuactors

Esta nueva técnica de accionamiento de las distintas partes del brazo, a diferencia de otras más tradicionales, introduce un resorte entre los elementos motor o de engrane y la salida del actuador, lo cual permite una mayor estabilidad en el control de la fuerza y protección ante cargas de choque.

5.3.2.3 Sensores

Los brazos están provistos de múltiples sensores útiles para la realización de diferentes tareas, como una cámara en la “mano” del robot (figura 5-6), sensores infrarrojos, sensores de presión en la muñeca, acelerómetros y diferentes elementos de entrada para el usuario, como los botones que se pueden observar en la zona de la muñeca de las figuras 5-4 y 5-6.

Figura 5-6. Sensores de la parte inferior de la muñeca del robot.

6 ANTECEDENTES

El proyecto europeo DREAM (http://www.robotsthatdream.eu) se desarrolló entre los años 2016 y 2019 e implicó el trabajo coordinado de 5 universidades europeas. El objetivo del proyecto fue la creación de una arquitectura cognitiva para un robot que soportase los principios de CDR, y que incluyese procesos propios del sueño en humanos. Estos procesos se basan en que, durante las etapas de sueño, el cerebro consolida la información adquirida creando generalizaciones de modelos y asentando el conocimiento.

La arquitectura cognitiva utilizada en el proyecto DREAM fue la desarrollada por el GII, Multilevel Darwinist Brain (MDB), que fue mejorada en numerosos aspectos. En el marco de este TFG, nos centraremos únicamente en uno de los elementos de esta arquitectura, el sistema de aprendizaje de modelos, que se describe con detalle en el apartado siguiente.

6.1 Multilevel Darwinist Brain

El MDB (Multilevel Darwinist Brain) es una arquitectura o mecanismo cognitivo que permite a un agente autónomo decidir las acciones que debe aplicar en su entorno de cara a satisfacer sus motivaciones. El mecanismo se basa en una serie de teorías bio-psicológicas [12,13,14] que relacionan el cerebro y su funcionamiento mediante un proceso Darwinista. El MDB se basa en un modelo cognitivo utilitario que se describe a continuación.

A la hora de construir un modelo cognitivo para un agente autónomo, es necesario establecer las condiciones en que se produce la interacción de dicho agente su entorno:

- Para la realización de cualquier tarea debe existir, en primer lugar, una motivación que guíe el comportamiento en función del grado de satisfacción de esta. Así, definimos la motivación como:

  Necesidad o deseo que lleva al agente a actuar

  Todo agente debe poseer una motivación implícita que le lleva a actuar y que puede variar con el paso del tiempo.
  Se asume la existencia de un agente que posee sensores independientes que le permiten obtener información sensorial de su entorno y de sí mismo, y también actuadores que le permiten ejecutar acciones.
- Definimos una *acción* A(t) de un agente como la activación de un actuador en un instante de tiempo t.
- La *percepción externa* e(t) de un agente está compuesta por la información sensorial que es capaz de adquirir del entorno en el que desarrolla su actividad a través de sus sensores externos.
- La *percepción interna* i(t) de un agente está compuesta por la información sensorial que posee sobre sus sensaciones internas.
- Definimos la *percepción global* G(t) del agente como la formada tanto por la *percepción externa* e(t) como por la *percepción interna* i(t).
- La *satisfacción* s(t) establece el grado de cumplimiento de la motivación. Se define como una función S de la percepción global, y por tanto, de las percepciones externa e interna:

$$s(t) = S[G(t)] = S[s(t), i(t)]$$

- Las percepciones externa e interna se pueden relacionar con la última acción realizada por el agente A(t-1), con las percepciones externa e interna en el instante de tiempo anterior e(t-1) e i(t-1), y con los eventos externos e internos

no controlados por el agente de tiempo característico inferior al modelable a través de las percepciones X_e(t-1) y X_i(t-1):

$$e(t) = W[e(t-1), A(t-1), X_e(t-1)] \quad ; \quad i(t) = I[i(t-1), A(t-1), X_i(t-1)]$$

Si, como primera aproximación, despreciamos los eventos no controlados X, tenemos que la satisfacción del agente resulta:

$$s(t) = S[e(t), i(t)] = S\{[e(t-1), A(t-1)], [i(t-1), A(t-1)]\}$$

La interacción del agente con su entorno le debe llevar a la satisfacción de la motivación que, sin pérdida de generalidad, se puede expresar como la maximización de la función de satisfacción, es decir:

$$max\,[s(t)] = max\,(S\,\{W\,[e(t-1), A(t-1)], I\,[i(t-1), A(t-1)\,]\,\})$$

La única variable susceptible de ser modificada en el proceso de maximización es la acción, ya que en general, la percepción externa y la percepción interna no se pueden manipular.

Como consecuencia, un mecanismo cognitivo debe explorar el espacio de acciones posibles para maximizar la función de satisfacción. Esto implica la utilización de un algoritmo de búsqueda de extremos sobre la función S que, a su vez, depende de W e I. Previamente a poder llevar a cabo este proceso, debemos encontrar la función S y, por tanto, también las funciones W e I, aplicando de nuevo un algoritmo de búsqueda de extremos, pero, en este caso, sobre un espacio de funciones. En este segundo proceso de búsqueda se trata de maximizar la similitud de las funciones I, W y S con las funciones reales que representan el estado interno del agente, el entorno y la función de satisfacción del agente. Es decir, tratamos de modelar la realidad del agente.

Tendremos así cuatro algoritmos de búsqueda de extremos, tres en paralelo para encontrar las mejores funciones W, I, y S y un último que utiliza estas funciones para hallar la mejor acción.

En base al modelo cognitivo anterior, podemos concluir que, en general, se busca una función que relacione la percepción externa e e interna i con la satisfacción S en los instantes t y t+1 tras aplicar una acción individual A. Dado que tratar de obtener un modelo tan general resulta una tarea inabordable por su complejidad, se ha optado por simplificar el problema dividiendo este modelo en dos partes diferenciadas: por un lado los modelos de mundo e interno y por otro, el modelo de satisfacción.

- **Modelo de mundo:** Función que permite evaluar las consecuencias de la ejecución de una acción en el entorno. Es decir, dadas las m entradas sensoriales $e(t)_1$, $e(t)_2$, ..., $e(t)_m$ y una acción $A(t)$ aplicada en el instante de tiempo t, nos proporciona los valores sensoriales en el instante de tiempo siguiente:
  $e\,(t+1)_1$, $e\,(t+1)_2$, ..., $e\,(t+1)_m$.

Como vemos, coincide con la definición de W que veíamos anteriormente.

- **Modelo interno:** Función que permite evaluar las consecuencias de la ejecución de una acción en las sensaciones internas del agente. Es decir, dadas las m entradas sensoriales $i(t)_1$, $i(t)_2$, ..., $i(t)_m$ y una acción $A(t)$ aplicada en el instante de tiempo t, nos proporciona los valores sensoriales internos en el instante de tiempo siguiente $i\,(t+1)_1$, $i\,(t+1)_2$, ..., $i\,(t+1)_m$.
  El modelo interno coincide con la definición de I del apartado anterior. Ejemplos de este tipo de sensores podrían ser un sensor de hambre, sed, estrés, etc. que trasladados a robots reales serían, por ejemplo, un sensor de nivel de batería, de

temperatura de CPU, es decir, sensaciones que no dependen del mundo exterior al agente y que influyen en su satisfacción.

- **Modelo de satisfacción:** Función que permite predecir la satisfacción $S(t)$ a partir de las entradas sensoriales en el instante t.
  Un modelo de satisfacción relaciona entradas sensoriales y satisfacción en el mismo instante de tiempo. En la práctica, los modelos de satisfacción nos proporcionarán la satisfacción predicha para el agente tras aplicar la acción.
  Además de los modelos, debemos definir una estructura de datos básica en el MDB, el par acción-percepción:

- **Par acción-percepción:** Conjunto de valores formado por las entradas sensoriales y la satisfacción relacionados con la aplicación de una acción en el entorno real (ver figura 6 para una representación esquemática). Constituyen la información fiable y real de la que dispone el mecanismo para tratar de obtener los modelos y se utiliza como patrón a la hora de perfeccionarlos. En una notación más biológica, los pares acción-percepción constituyen los episodios de la ”vida” del agente.

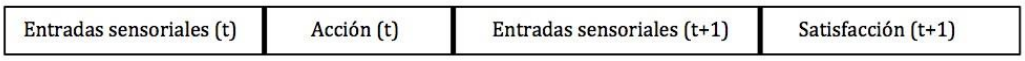

Figura 6-1. Par acción percepción.

Un esquema funcional básico que concreta el modelo cognitivo establecido en el apartado anterior se muestra en la figura 6-2 y se podría explicar como sigue:

El objetivo final del mecanismo cognitivo es obtener la acción que debe ejecutar el agente en su entorno en cada instante para satisfacer sus motivaciones. Esta acción se aplica al entorno a través de los actuadores obteniendo nuevos valores de sensorización para el agente. Así, tras cada interacción con el mundo real, tenemos un par acción-percepción que refleja la relación entradas-salidas, esto es, lo que percibió y actuó el agente y a qué percepción le ha llevado, que es lo que los modelos deben predecir.

Estos pares acción-percepción se almacenan en una memoria a corto plazo (MCP) y a continuación comienzan los procesos de búsqueda de los modelos de mundo, modelos internos y modelos de satisfacción que mejor predigan los contenidos de la MCP. Estos tres procesos son concurrentes. Tras su finalización, los mejores modelos de cada tipo se marcan como modelo de mundo actual, modelo interno actual y modelo de satisfacción actual y se utilizan en el proceso de optimización de la acción para determinar su efecto sobre la satisfacción. Así, la acción se escoge en un proceso interno al MDB, donde los modelos actuales representan la ”realidad” en cada instante de tiempo. Los modelos representan el conocimiento del entorno y de sí mismo que posee el agente a partir de la experiencia pasada.

La acción seleccionada se aplica en el entorno a través de los actuadores obteniendo nuevos valores de sensorización y así nuevos pares acción-percepción que se almacenan en la MCP. Este ciclo básico se repite (lo denominamos una iteración del mecanismo) y, con el paso del tiempo, la información real disponible es mayor y, en consecuencia, los modelos son más precisos y las acciones escogidas a partir de ellos más adecuadas.

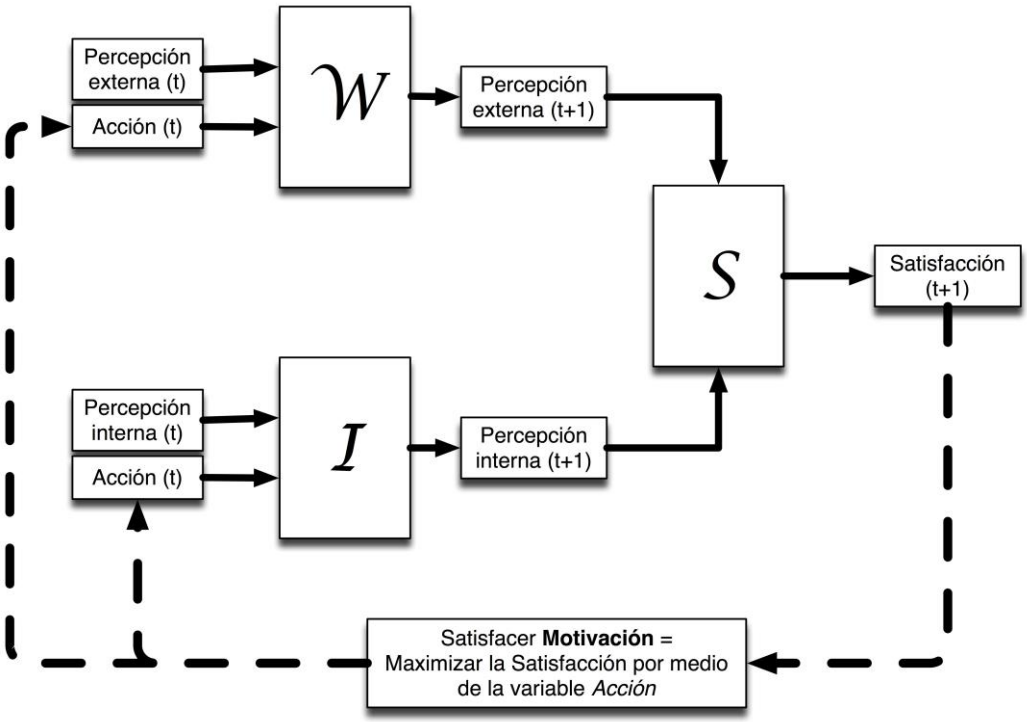

Figura 6-2. Esquema funcional del modelo cognitivo.

Los procesos de búsqueda de los modelos representan el modo en el cual el MDB adquiere el conocimiento a partir de los hechos y son procesos de aprendizaje, no de optimización (buscamos la mejor generalización posible a lo largo del tiempo, no queremos minimizar una función de error en un instante dado). Por este motivo, la técnica de búsqueda que utilicemos debería permitir incorporar nueva información al dominio del problema.

En la elección de la representación matemática de los modelos y de la técnica de búsqueda reside la inspiración tomada de las teorías bio-psicológicas que fundamentan el MDB [12, 13, 14]. De acuerdo con estas teorías, el desarrollo mental de un organismo sigue un proceso darwinista y es la interacción con el entorno la que va moldeando el cerebro mediante la activación o desactivación de ciertas conexiones cerebrales. En consecuencia, en el MDB los modelos están representados por Redes Neuronales Artificiales y los procesos de aprendizaje que implementan el proceso darwinista son Algoritmos Evolutivos (AE).

Los AE permiten un proceso gradual de aprendizaje si controlamos el número de generaciones de evolución en cada iteración. Así, los modelos obtenidos en un instante dado para un contenido de la memoria a corto plazo son fácilmente ajustables para otro contenido posterior, es decir, nunca estarán excesivamente adaptados a una situación particular (necesario asumiendo que el entorno es dinámico). Para lograr este aprendizaje gradual, debemos mantener las poblaciones de modelos entre iteraciones del MDB, de modo que se puedan seguir optimizando según los cambios y únicamente al comienzo de la ejecución del mecanismo los modelos serán aleatorios.

La función de calidad de los modelos se obtiene como una función de error de las salidas que proporcionan las redes que representan los modelos cuando se prueban todos los pares acción-percepción que están almacenados en la MCP en una cierta iteración.

El aprendizaje de los modelos de mundo a través de algoritmos evolutivos (AEs) resultó satisfactorio en ciertos problemas en tiempo real, pero el coste computacional era más alto

que el deseable. El auge de los modelos de *Deep Learning* hace que los investigadores del GII se planteen el uso de algoritmos no basados en evolución para el aprendizaje en tiempo real de esos modelos, tal y como se estudiará en el apartado siguiente. En concreto, en este TFG se desarrollará un procedimiento para el aprendizaje on-line de modelos de mundo mediante algoritmos de *Deep Learning*. Este procedimiento se basa en la gestión adecuada de la memoria y los pasos de aprendizaje para hacerlo factible en tiempo real.

7 DESARROLLO

En este apartado se detallarán las principales contribuciones de este TFG. En primer lugar, se explicará el procedimiento de aprendizaje propuesto, a continuación, se describirá el montaje experimental utilizado para la obtención del conjunto de datos necesario para las pruebas. Después se explicará la implementación de los algoritmos propuestos en Keras y finalmente se describirán las pruebas realizadas y los resultados obtenidos.

Para el desarrollo del modelo se ha utilizado la herramienta *TensorFlow* en lenguaje de programación de Python.

7.1 Procedimiento de aprendizaje on-line

Tal y como se explicó en el apartado de antecedentes, el aprendizaje *on-line* del MDB se ha basado, hasta ahora, en la utilización de una memoria a corto plazo en la que se iban guardando los pares acción-percepción. Este método ha resultado adecuado y se ha probado en numerosos casos reales, pero los algoritmos de aprendizaje *on-line* como el *Adam*, se basan en otras estrategias de uso de la memoria, que se deben explorar.

En concreto, a continuación, se detallará el funcionamiento de este algoritmo cuando se utiliza un descenso de gradiente estocástico o por mini-lotes, y se comentará por qué su uso en el MDB no es válido. Después se explicará con detalle el funcionamiento de la estrategia típica del MDB, denominada *sliding window* y se propondrá una combinación de ambas, con el objetivo de establecer cuál es el procedimiento de aprendizaje más adecuado para el uso del algoritmo *Adam* en el MDB.

7.1.1 Algoritmo estocástico

Tradicionalmente, los algoritmos estocásticos son los algoritmos por defecto cuando se habla del aprendizaje redes neuronales [8]. Su funcionamiento es bastante simple, tal y como se ha comentado en los fundamentos teóricos.

El algoritmo estocástico trabaja de tal manera que, con cada iteración del robot en el mundo, se incorpora a la memoria una nueva muestra. Una vez introducida, el algoritmo calcula el error de predicción de dicha muestra y actualiza el valor de los pesos de la red. A continuación, se reinicia este proceso, y una nueva muestra es incorporada en la memoria, reemplazando a la anterior.

En la siguiente imagen se puede observar gráficamente el funcionamiento del algoritmo adaptado al MDB. Suponemos que el cuadrado azul claro es la Memoria a Corto Plazo de tamaño unitario y que cada “dato” se corresponde con un par acción-percepción. En cada iteración el algoritmo entrena con el dato marcado para desecharlo a continuación, como marcan las flechas.

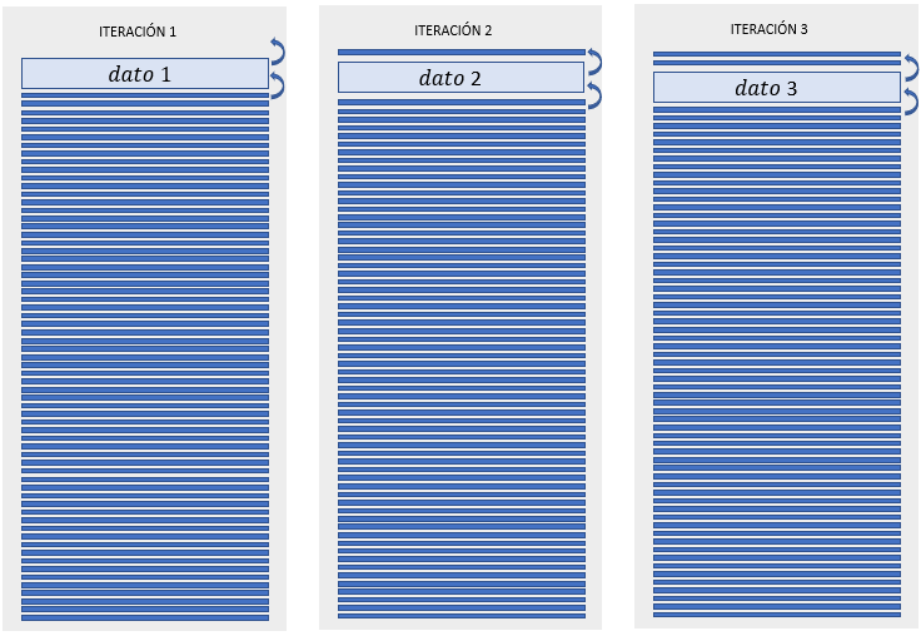

Figura 7-1. Esquema de funcionamiento de un algoritmo estocástico.

Aunque los algoritmos estocásticos sí que permiten un aprendizaje en tiempo real, hay un motivo principal por el que se ha descartado su uso en este trabajo. Al hacer una actualización con cada dato y descartar los anteriores, se hace un uso escaso de la información obtenida, lo cual ralentiza el aprendizaje. Esta estrategia puede ser efectiva en ciertos ámbitos, pero no en la robótica cognitiva, donde se requiere una respuesta eficiente en tiempo real. Además, este aprendizaje tan gradual hace que el modelo cambie continuamente, de modo que el funcionamiento del MDB no es estable.

7.1.2 Algoritmo con mini-lotes

Cualquier algoritmo de descenso de gradiente por mini-lotes supone un barrido completo del conjunto de datos, pero agrupando la totalidad de datos en subconjuntos más pequeños que no se solapa, y realimentando el error de cada uno de ellos. El caso más extremo sería un algoritmo por mini-lotes de tamaño unitario, que se correspondería con el funcionamiento de un algoritmo estocástico.

Para entender fácilmente el funcionamiento de este esquema en MDB, suponemos un conjunto de datos de tamaño cien y tamaño de mini-lote de cinco. Inicialmente se tiene que esperar la entrada de cinco pares acción-percepción para poder trabajar con ellos. Con cada nueva iteración se eliminan aquellos datos que estaban almacenados en el mini-lote en la iteración anterior y se vuelve a esperar otra vez la entrada de otros cinco pares de datos para poder seguir con el entrenamiento. En la figura 7-2 de la siguiente página se explica visualmente el funcionamiento de la memoria de un algoritmo que utilice mini-lotes suponiendo un tamaño del mini-lote de cinco.

Aplicado a nuestro problema de robótica cognitiva se puede concluir que la utilización de este tipo de algoritmo no es realmente útil, ya que no permite el aprendizaje en cada iteración del robot. Se tendrían que esperar *n* nuevos pares de datos acción-percepción, siendo *n* el número que indica el tamaño del mini-lote, antes de poder volver a entrenar el modelo, lo cual en una operación real no es factible dado el tiempo que puede implicar realizar una acción con el robot.

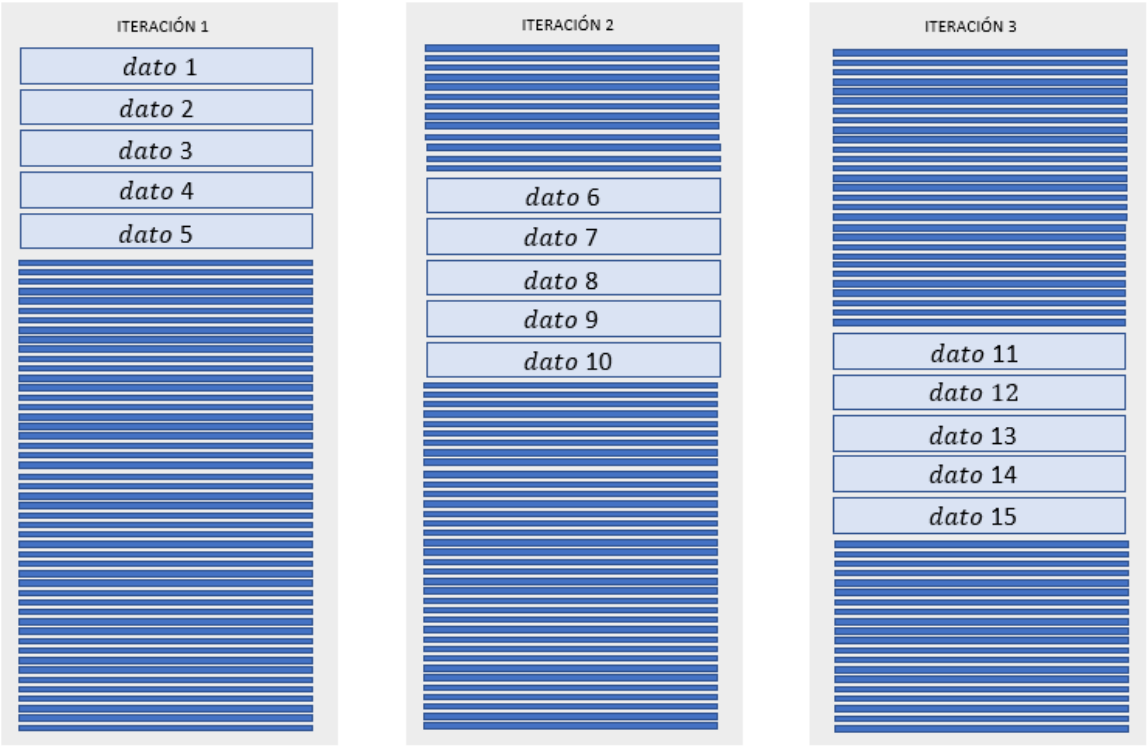

Figura 7-2. Esquema de funcionamiento de un algoritmo con mini-lotes

7.1.3 Algoritmo con una ventana deslizante.

Una variante con respecto a las dos anteriores propuestas es utilizar una ventana deslizante, en inglés *sliding window*, que es la que se venía utilizando en el MDB para la gestión de la memoria a corto plazo [7]. La estrategia de actualización de la memoria en esta aproximación sigue una estrategia FIFO, del inglés *First In, First Out*. Su funcionamiento es bastante sencillo (ver Figura 7-3): suponemos un conjunto de 100 datos y suponemos también una ventana que va a tener una capacidad de almacenamiento para 10 datos. Para una iteración $i = 0$, se tienen en la ventana de la memoria los diez primeros datos que han llegado. El resto de los rectángulos azules más estrechos, como ocurre en las explicaciones anteriores, representan el resto del conjunto de datos.

La ventana deslizante comenzará almacenando los diez primeros datos y en las iteraciones posteriores irá barriendo el conjunto de datos, incorporando en cada iteración un par acción-percepción nuevo y eliminando el más antiguo. De esta forma, el algoritmo ve el mismo dato diez veces, o el número equivalente a la capacidad de la ventana que se proponga.

En la siguiente figura se puede observar el funcionamiento:

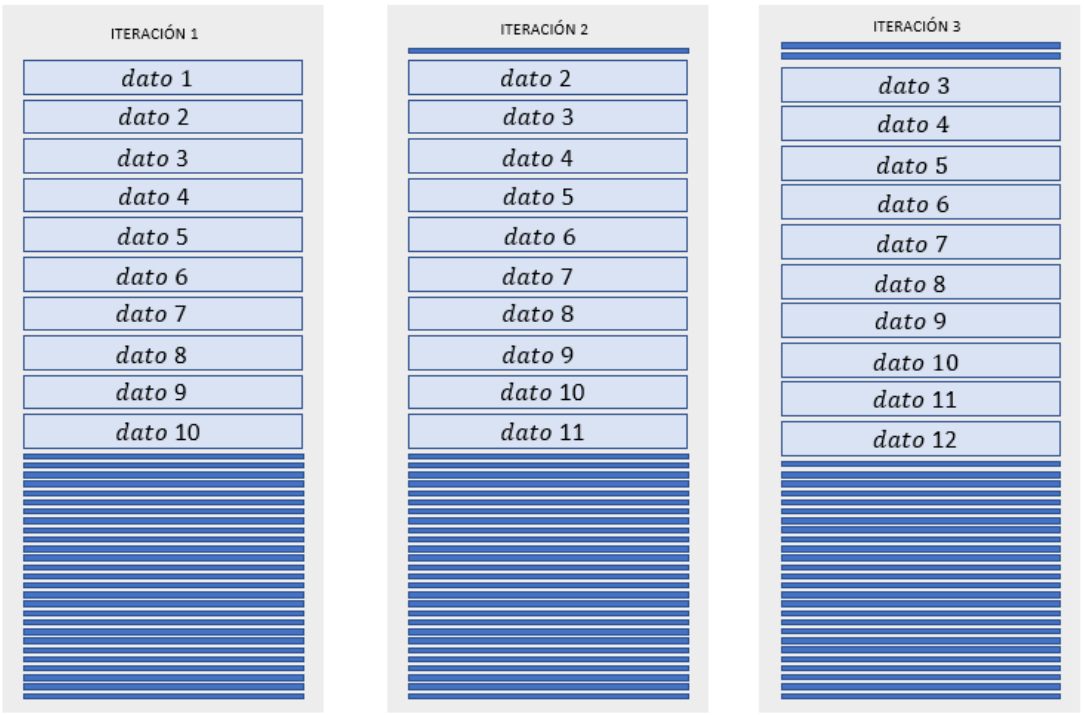

Figura 7-3. Esquema de funcionamiento de la Sliding Window.

Ante la llegada de otro par acción-percepción, representado en la figura como “dato”, el primer dato de la ventana de tamaño diez es eliminado, mientras que ahora, en última posición aparecerá uno nuevo. Se tiene entonces que, para la iteración $i = 1$, la ventana de la memoria tendrá el mismo número de datos, pero los datos serán, en un 10%, diferentes, tal y como se muestra en la anterior figura.

A medida que van entrando nuevos datos, los datos contenidos en la ventana se van actualizando hasta llegar a la última iteración:

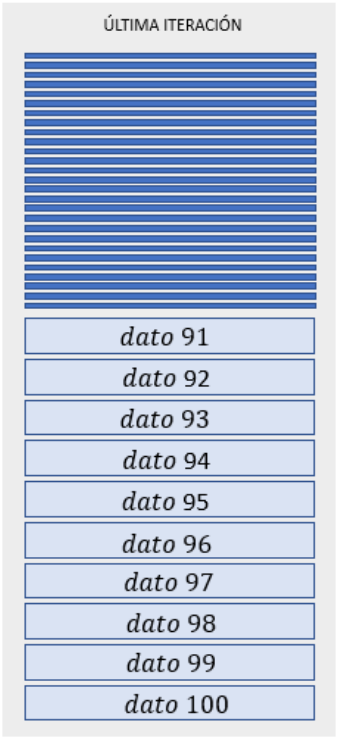

Figura 7-4. Última iteración del proceso de la ventana deslizante.

Una vez conocido el funcionamiento de la ventana deslizante, se puede afirmar que, entre las opciones presentadas, esta sigue pareciendo la más adecuada para el aprendizaje *on-line* desarrollado en este trabajo. La decisión de utilizar esta estrategia para el tratamiento de datos se basa en que permite el aprendizaje en tiempo real, al contrario de lo que pasa con el entrenamiento usando mini-lotes de datos. Aunque bien es cierto que inicialmente se debe esperar el tiempo necesario para que se complete la capacidad de datos de la ventana, en las posteriores iteraciones, con la llegada de un dato, la ventana se actualiza e inmediatamente se entrena con ese nuevo conjunto de datos, sin esperas adicionales. De esta forma se potencia el uso de la información obtenida, que se mantiene en la memoria durante varias iteraciones, dependiendo del tamaño de la ventana.

7.1.4 Nueva propuesta de memoria

Uno de los retos más importantes de este trabajo era el manejo de la memoria de datos para el aprendizaje online. Por este motivo se propone un nuevo sistema de tratamiento de datos para el aprendizaje: la combinación de un algoritmo estocástico o un algoritmo por mini-lotes y el uso de una ventana deslizante que permita el barrido del conjunto de datos. Este nuevo tratamiento de la memoria consiste principalmente en el uso de una ventana deslizante de una capacidad *n* de datos.

Una vez que se tiene la ventana deslizante configurada se propone la utilización de una lectura de datos dentro de esa ventana, bien estocástica o bien a través de mini-lotes. De esta forma, se pretende analizar si el uso habitual de los datos en *Deep Learning on-line* combinado con la ventana deslizante que se ha venido usando en el MDB, mejora los resultados del aprendizaje. A continuación, en la figura 7-5, se muestra gráficamente cómo se realiza este proceso.

- Lectura de datos estocástica: si se observa la figura siguiente se puede entender el procedimiento de forma sencilla. En primer lugar, se tiene una ventana deslizante de tamaño 5 que barre el conjunto de datos. Esos cinco datos, a su vez, se utilizarán para entrenar uno por uno, a través del algoritmo de optimización propuesto. Es decir, si se aísla la ventana (figura 7-5), esos datos se entrenarán como si se tratase de un algoritmo de descenso de gradiente estocástico, actualizando los parámetros de la red con cada dato.

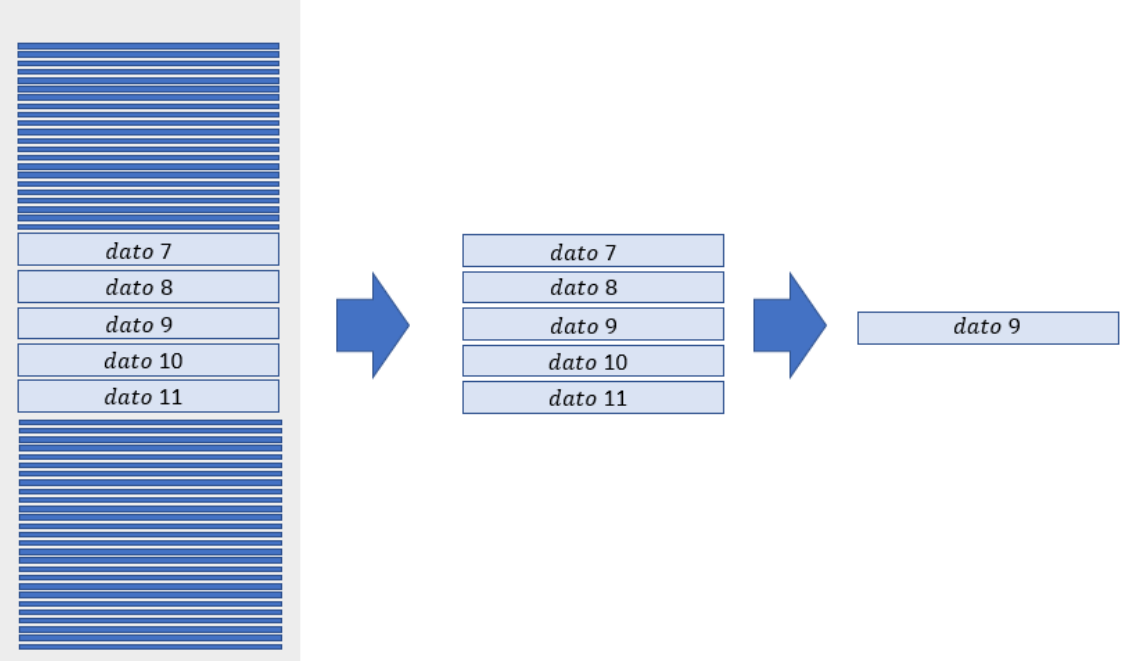

Figura 7-5. Sliding Window con estocástico.

- Lectura de datos con mini-lotes: la idea viene a ser muy parecida a la que se presentaba en el punto anterior, sin embargo, los datos no se entrenarán uno por uno dentro de una iteración *i*, sino que se entrenarán siguiendo el funcionamiento de un algoritmo de descenso de gradiente por mini-lotes.

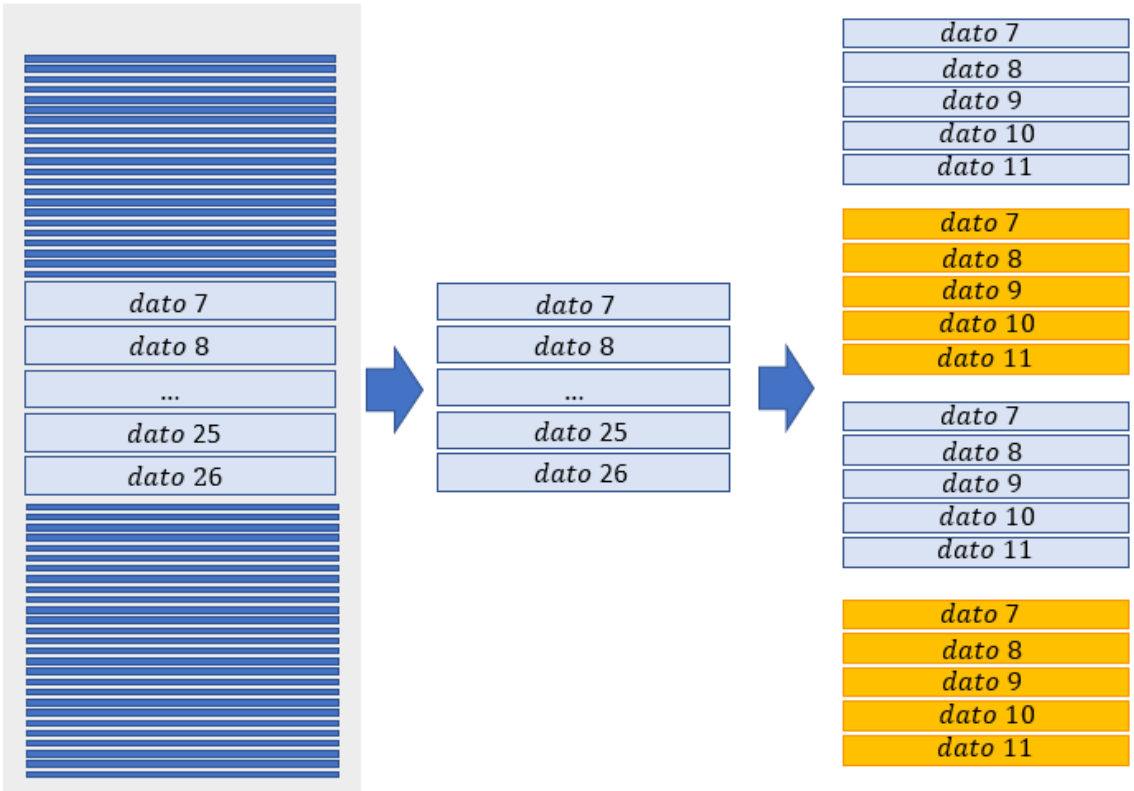

Figura 7-6. Sliding Window con mini-lotes.

En la anterior figura se puede ver que, una ventana con capacidad para veinte datos se divide a su vez en mini-lotes de tamaño cinco. Esta ventana de veinte se entrenará como si se tratase de un conjunto de datos para un algoritmo que funciona a través de mini-lotes. En la siguiente iteración, la ventana de 20 se actualiza desechando primer valor incorporado e introduciendo uno nuevo. Y se volverá a entrenar como lo hizo en la iteración anterior.

7.2 Programación con Keras

El uso de la API de Keras de TensorFlow ha facilitado enormemente el trabajo y la consecución del programa para realizar los diferentes entrenamientos gracias a la simplicidad de uso de las diferentes funciones implementadas previamente en diferentes librerías. Aun así, diferentes tareas como la programación del funcionamiento de la memoria, así como la representación de gráficas se han realizado usando las funciones y librerías básicas del Python. El algoritmo de entrenamiento utilizado es el *Adam Optimizer*. En este trabajo no se ha modificado los parámetros básicos del para mejoras de resultados en las pruebas, así que se han mantenido los valores por defecto del programa.

7.2.1 Programación de la red neuronal

Para este TFG los modelos de aprendizaje serán construidos como una red neuronal perceptrón multicapa con diferentes capas ocultas formadas por un número determinado de neuronas cada una de ellas.

En Keras, el modelo más básico se puede diseñar apilando capas de neuronas, tal y como se muestra a continuación. Este sencillo modelo se llama *Sequential*.

```
def baseline_model():
    model = Sequential()
    model.add(Dense(10, input_dim=7 , kernel_initializer=normal, bias_initializer='ones', activation='tanh'))
    model.add(Dense(10, kernel_initializer=normal, bias_initializer='ones', activation='tanh'))
    model.add(Dense(3, kernel_initializer=normal, bias_initializer='ones', activation='tanh'))
    # Compile model
    model.compile(loss='mse', optimizer='adam' , metrics=['mse'])
    return model
```

Figura 7-7. Programación de una red de neuronas con Keras.

Aunque más tarde se hablará de cómo se ha decidido la configuración de la red neuronal, se puede comentar brevemente la figura anterior.

El proceso de diseño del modelo es tan sencillo como seleccionar el tipo de modelo, en este caso el *Sequential*, y comenzar a añadir capas. Primero se crea la primera capa de la red, como se puede observar en el ejemplo de la Figura 7-7, que debe coincidir con el número de entradas de datos, después múltiples capas ocultas (en la figura 7-7 las dos capas son de 10 neuronas) y finalmente, una última capa que tiene que coincidir con el número de salidas del modelo.

La función de activación utilizada es la tangente hiperbólica. En muchas ocasiones se utiliza la función ReLU, pero la presencia de datos con valores negativos impedía el correcto aprendizaje del conjunto de datos. Hay que recordar que la función ReLU está restringida por lo siguiente: cuando los valores en el eje de abscisas descienden por debajo de cero, el valor de la función toma el valor de cero. Debido a esto, en nuestro caso la aproximación no es correcta para valores negativos.

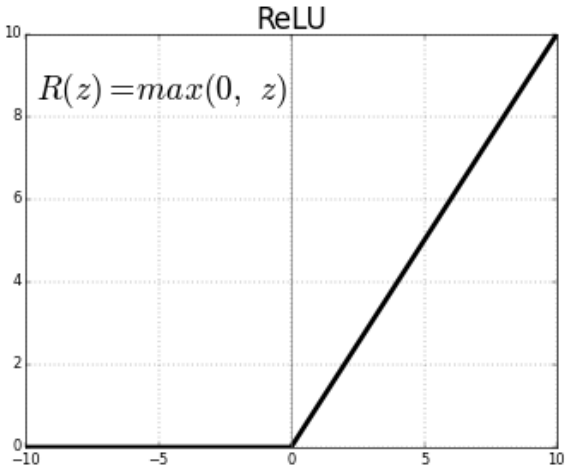

Figura 7-8. Función de activación ReLU.

7.2.2 Programación de la ventana deslizante

El diseño de la ventana deslizante se ha implementado directamente en el lenguaje de Python. Consiste en una función que contiene un bucle sencillo que va recorriendo el conjunto de datos de entrenamiento y devuelve los datos con una ventana de tamaño predefinido (100 en el siguiente ejemplo). Se podría escribir el pseudocódigo como:

Inicio

$X_{ventana} = [\,]$ /*Se define un vector x*/

$Y_{ventana} = [\,]$ /*Se define un vector y */

X_{train} /*vector de entradas del conjunto de entrenamiento*/

Y_{train} /*vector de salidas del conjunto de entrenamiento*/

Desde $contador = 0$ hasta leer todo el conjunto de entrenamiento hacer:

$X_{ventana} = X_{train}\,[contador: 100 + contador]$

$Y_{ventana} = Y_{train}\,[contador: 100 + contador]$

$contador = contador + 1$

Se devuelve el valor de $X_{ventana}$ e $Y_{ventana}$

Una vez son devueltos los valores de x e y contenidos en la ventana de datos se procede al entrenamiento. Hasta que se complete el barrido del conjunto de datos se repetirá el bucle que controla la variable "*contador*" que se puede ver en el anterior pseudocódigo.

7.2.3 Programación del nuevo sistema de gestión de memoria

Para el nuevo sistema de gestión de memoria a corto plazo se ha introducido en el código básico de la ventada deslizante un bucle que relea la ventana de datos y entrene el modelo tal y como se ha explicado en el apartado 7.1.4. Se puede mostrar el pseudocódigo de la siguiente manera:

Inicio

$X_{ventana}$ /*vector de entradas de la ventana deslizante*/

$Y_{ventana}$ /*vector de salidas de la ventana deslizante*/

Desde $i = 0$ hasta releer todo el conjunto de entrenamiento, hacer:

Entrenamiento con parámetros ($X = X_{ventana}$, $Y = Y_{ventana}$, $tamaño\ de\ lote$)

$contador = contador + 1$

Si se quiere ejecutar el nuevo sistema de gestión de memoria para un algoritmo estocástico, simplemente se puede indicar un tamaño de lote unitario. Para un algoritmo con mini-lotes se debe señalar el tamaño seleccionado.

7.2.4 Programación de las funciones de entrenamiento

Las funciones de entrenamiento del conjunto de datos están implementadas directamente en la API de Keras. Las dos funciones disponibles son *fit* y *train_on_batch*, de las cuales se hablará brevemente a continuación:

- *model.fit()*

```
fit(x, y, batch_size, epochs, callbacks)
```

x e *y* son los conjuntos de entrenamiento (x las entradas e y las salidas). En el siguiente apartado se explicará cómo se ha dividido el conjunto de datos, pero del total del conjunto se ha dividido en dos, uno de entrenamiento y otro de test.

El tamaño de lote (*batch_size*) es el parámetro que permite dividir el conjunto de entrenamiento en mini-lote para entrenar.

El número de épocas de entrenamiento (*epoch)* es el número de pasos para entrenar el modelo. Para este trabajo la especificación del número de épocas de entrenamiento caso no es muy relevante, ya que se ha utilizado una función (*callbacks*) para parar el entrenamiento. Esta actúa de tal manera que en caso de que el error de entrenamiento se estanque o vuelva a subir se para el proceso de aprendizaje para evitar el sobreajuste (*overfitting*). La función seleccionada es la *keras.EarlyStopping*().

- *model.train_on_batch()*

```
train_on_batch(x, y, sample_weight=None, class_weight=None)
```

Esta función tiene el mismo objetivo que la anterior, entrenar datos. Sin embargo, a diferencia del *model.fit()*, el conjunto de datos de entrenamiento es introducido manualmente y solo realiza una actualización de los pesos de la red con cada llamada.

7.2.5 Evaluación del modelo

El conjunto de datos utilizados para la evaluación del modelo de red ha sido el conjunto de test. Por lo general, del conjunto de datos total, se utiliza un porcentaje menor para el conjunto de test que para el conjunto de entrenamiento (10%, 20% o 30%). Para la obtención de un resultado en la evaluación se ha calculado error en valor absoluto entre las predicciones y las salidas. Se utiliza el valor de test para evaluar las salidas de la red, ya que al ser datos diferentes a los de entrenamiento se prueba de mejor manera la eficacia del proceso de aprendizaje.

El valor de las predicciones se almacena en un vector utilizando la siguiente función:

```
predict(x, batch_size=None, verbose=0, steps=None, callbacks=None)
```

Como se ha comentado antes, x será *x_test* y el resto de los valores se tomarán por defecto. Aunque Keras tiene una función implementada llamada *evaluate()*, en este trabajo se ha optado para calcular directamente el error de la siguiente forma, donde $salida\_test$ hace referencia a la salida del conjunto de test:

$$error = | salida\_test - predicciones |$$

Debido a las características del funcionamiento del robot comentadas con anterioridad, las predicciones no deben tener un error mayor de cinco milímetros con respecto a la salida original.

8 PRUEBAS REALIZADAS

En este apartado se explicarán las diferentes pruebas realizadas a lo largo de todo el desarrollo del trabajo y los resultados obtenidos serán analizados mediante gráficas y tablas que nos permitan comparar de manera sencilla los errores obtenidos con cada aproximación. Para ello se pretenderá estudiar dos aspectos clave del problema propuesto: la tipología de la red y el tamaño de la ventana deslizante para el aprendizaje on-line.

8.1 Desarrollo de las pruebas

En este apartado se describirá la metodología experimental utilizada en este trabajo para a continuación comentar los resultados y elaborar una conclusión basada en los datos obtenidos.

A la hora de analizar un sistema de aprendizaje basado en *Deep Learning*, hay numerosos parámetros a configurar que condicionan el resultado final. De entre todos ellos, los que hemos considerado más relevantes para este dominio de aplicación son: el número de iteraciones de aprendizaje, el tamaño de la red neuronal, el tamaño de la ventana deslizante, y el número de pasos de entrenamiento para cada lote. Estos 4 parámetros podrían ser barridos de forma exhaustiva para hallar la combinación ideal en el conjunto de datos proporcionado, pero hemos considerado más adecuado para este proyecto un análisis funcional de los mismos, es decir, utilizar una metodología basada en criterios obtenidos del dominio del problema a resolver, y así reducir el número de combinaciones a tener en cuenta.

8.1.1 Iteraciones de aprendizaje

De cara a estimar el número de iteraciones requeridas para aprender el conjunto de datos, se utilizará la función *EarlyStopping* de Keras, que se basa en un criterio dinámico de estabilidad en el error para llegar a un umbral predefinido. Esto nos permitirá tener una idea del tiempo que le lleva al algoritmo alcanzar un nivel de error inferior al límite de 5 mm establecido anteriormente como objetivo.

Esta función *EarlyStopping* no está disponible en el método *train_on_batch()* de Keras que se usará en las pruebas de este TFG, y solo funciona con el método *fit()*, por lo que esta primera prueba se realizará con dicho método. Esto implica utilizar el conjunto total de datos y seleccionar un tamaño de mini-lote para optimizar el aprendizaje. Como no tenemos criterio para fijar este tamaño de mini-lote a priori, se utilizarán valores no muy grandes, asumibles luego en las comparaciones con el método train_on_batch(), en concreto, entre 10 y 100.

Por lo tanto, se puede decir que, para suplir la deficiencia en el método *train_on_batch* de la función utilizada con el método *fit* que actúe como criterio de parada se ha realizado la siguiente hipótesis. Tanto con el funcionamiento de la función *fit* como con la función *train_on_batch* se deben de obtener resultados muy similares después de cada lanzamiento del programa así que se ha supuesto que el número de iteraciones de entrenamiento debería ser equiparable para llegar a un error de test válido. Así que se saca la siguiente relación:

$$Iteraciones = épocas * \frac{tamaño_{datos_{entrenamiento}}}{tamaño_{lote}}$$

Para asegurar que el proceso sea estadísticamente neutro se realiza una validación cruzada con un *k-fold*. Esto consiste en separar el conjunto de datos en *k* partes. El entrenamiento se realiza *k* veces. En cada lanzamiento del programa, una de las *k* partes del conjunto se utiliza como conjunto de test y el resto como conjunto de entrenamiento. Para el siguiente lanzamiento se escoge otra de las *k* partes como conjunto de test y el resto como entrenamiento. Así hasta llegar a los *k* lanzamientos del programa.

Una vez obtenidas las iteraciones de entrenamiento proporcionadas por el método *EarlyStopping*, se comprobará si estas son adecuadas para el entrenamiento con train_on_batch() y una ventana deslizante.

Para ello se seleccionarán un conjunto de redes diferentes, con 4 tamaños de lote cada una, y se analizarán los niveles de error que se alcanzan.

8.1.2 Tamaño de red y ventana

En esta segunda prueba, una vez fijadas las iteraciones de aprendizaje, se pretende realizar una primera estimación del tamaño de red y de ventana adecuados para el conjunto de datos seleccionado. Para ello, se realizarán lanzamientos del programa con múltiples tipologías de red, cuantas más y más variadas, mejor, y cuatro tamaños de ventana deslizante y en función de los resultados se seleccionarán las combinaciones que den mejores valores de error de entrenamiento.

Los resultados que se quieren examinar en esta prueba se centran en los errores de test obtenidos después de entrenar el modelo, que se calculan como se ha comentado previamente en el apartado (7.2.5). Como se ha comentado anteriormente, el error de test máximo que se pretende cometer es de cinco milímetros ya que el *gripper* del robot real puede llegar a cometer ese error como máximo y se pretende generar un modelo de mundo lo más fiel posible al robot real.

Tras seleccionar las redes y tamaños más adecuados en un primer análisis se continúa con un segundo estudio mucho más exhaustivo que se centre en hallar la mejor combinación de estos dos parámetros. Para escoger los parámetros finales se prestará especial atención a los resultados obtenidos para el error de test, para la desviación del error, para el tiempo de cómputo de cada paso y del total y también a la complejidad de la red propuesta.

Manteniendo el código para el entrenamiento con ventana deslizante, cada una de las posibles combinaciones se ensayarán diez veces, todas ellas barajando el conjunto de datos de entrenamiento para intentar simular una validación cruzada con un *10-fold*, pero manteniendo invariable el conjunto de test. Los resultados se van a analizar realizando el promedio de todos los lanzamientos para cada combinación.

Con los resultados obtenidos se decidirá la tipología final de la red y el tamaño definitivo de la ventana de datos.

8.1.3 Pasos de entrenamiento

Partiendo del número de iteraciones, tamaño de red, y tamaño de ventana establecidos en las pruebas anteriores, se va a comprobar cómo se comportaría el modelo si en vez de entrenar una única vez en cada paso con un dato diferente, se repite un número *n* de veces el entrenamiento con este mismo dato. Para la programación de esta función simplemente se introduce un bucle que repita el entrenamiento con el mismo dato el número n de veces. Al pasar dos veces los mismos datos se reduce el número de pasos de entrenamiento a la mitad. Al pasarlo cinco veces, los pasos de actualización de ventana se reducirán un factor de cinco y así sucesivamente. De esta manera nos aseguramos de que las condiciones de entrenamiento sean lo más similares posible para cada número de repeticiones.

Primeramente, se hará un lanzamiento del entrenamiento del modelo con cada *n* escogida y en función de los resultados tanto de error de test como de tiempo total se seleccionarán las mejores opciones.

8.1.4 Nueva gestión de memoria a corto plazo

Uno de los aspectos más interesantes de este Trabajo de Fin de Grado es la nueva utilización de la memoria a corto plazo para entrenar el modelo. Es un concepto bastante diferente al utilizado hasta ahora, ya que combina la utilización de una ventana deslizante sobre un conjunto de datos con el funcionamiento de un algoritmo estocástico o con mini-lotes. El funcionamiento detallado se puede consultar en el apartado 7.1.4.

Manteniendo los parámetros fijados en las pruebas anteriores, se proponen diferentes tamaños de mini-lotes para llevar a cabo la prueba con el código diseñado.

8.2 Montaje experimental

Para la ejecución de las pruebas que conforman este trabajo de fin de grado fue necesaria la realización de un experimento para tratar de generar un modelo de mundo que, a través de una entrada sensorial y una acción, proporcione otro estado sensorial. Al final del primer experimento se obtiene un conjunto de datos que se utilizará en los apartados posteriores para realizar entrenamiento del modelo y obtener los resultados de las pruebas descritas anteriormente.

Para el experimento se ha utilizado el robot Baxter. Más concretamente se han estudiado los ángulos y las posiciones del brazo, ya que el objetivo del aprendizaje es el conocimiento de la posición de los “dedos” (*gripper*) del robot en función de los datos de entrada, que son los ángulos de posición de las articulaciones del brazo con respecto a un punto de referencia que se sitúa en el centro del cuerpo de este. Este desafío, que tradicionalmente se podría resolver calculando la posición de los “dedos” del robot a través de los ángulos y las distancias de los brazos, ahora se pretende resolver usando una red neuronal.

Los elementos principales que han sido necesarios para la generación y recogida de datos han sido el robot Baxter, por supuesto, la mesa que se puede ver en la figura 8-1 y una cámara modelo *Asus xtion pro live*.

- Robot Baxter: se ha programado para realizar acciones de manera aleatoria. Los ángulos máximos y mínimos que pueden alcanzar cada articulación del brazo del robot y la altura de la mesa que se sitúa delante de él en la figura 8-1 son las restricciones que tiene el robot para realizar las acciones. En el apartado 5.3 se pueden ver con detalle las especificaciones del robot.
- Mesa: como bien se ha dicho en el apartado anterior restringe el movimiento de la mesa. Sus medidas son 180 cm de longitud, 80 cm de anchura y 87 cm de altura. La medida más importante es la altura ya que el robot puede chocar contra ella.
- Cámara *Asus Xtion Pro Live*: está situada en el techo del espacio donde se realiza el experimento y permite conocer de manera certera la posición real del *gripper* del robot. Aunque esta posición se puede hallar de manera analítica, pueden existir holguras en las articulaciones o algún fallo en los sensores del robot que conduzcan a errores, y por lo tanto a una recogida de datos incorrecta.

Para cada acción aleatoria el robot genera siete entradas de datos que se corresponden cada una con los siete grados de libertad que posee el brazo del mismo, tal y como aparece en la figura (8-1). Los grados de libertad son: S0 (hombro), S1 (hombro), E0 (codo), E1

(codo), W0 (muñeca), W1 (muñeca) y W2 (muñeca). La posición mostrada en la figura 8-1 se corresponde con los datos de entrada de la tabla 1, donde también se muestra la posición del *gripper* en coordenadas cartesianas *x, y, z,* que son las 3 salidas que tiene la red.

	RIGHT S0	RIGHT S1	RIGHT E0	RIGHT E1	RIGHT W0	RIGHT W1	RIGHT W2	GRIPPER X	GRIPPER Y	GRIPPER Z
0	0.044	-1.058	1.269	1.941	-0.607	1.075	-1.205	0.565	-0.170	0.143

Tabla 1. Datos de las entradas (en rojo) y salidas del experimento (en azul).

Las siete entradas representan la acción a realizar por el robot y las 3 salidas representan la posición del *gripper* tras realizar la acción. Este modelo de mundo es un tanto particular, ya que aunque en la mayoría de modelos existen también entradas sensoriales, en este no. Las anteriores entradas y salidas de datos están configuradas para el brazo derecho del robot como bien aparece en el título de cada columna. A continuación, se puede ver la posición del robot real correspondiente a los anteriores datos, con cada articulación señalada.

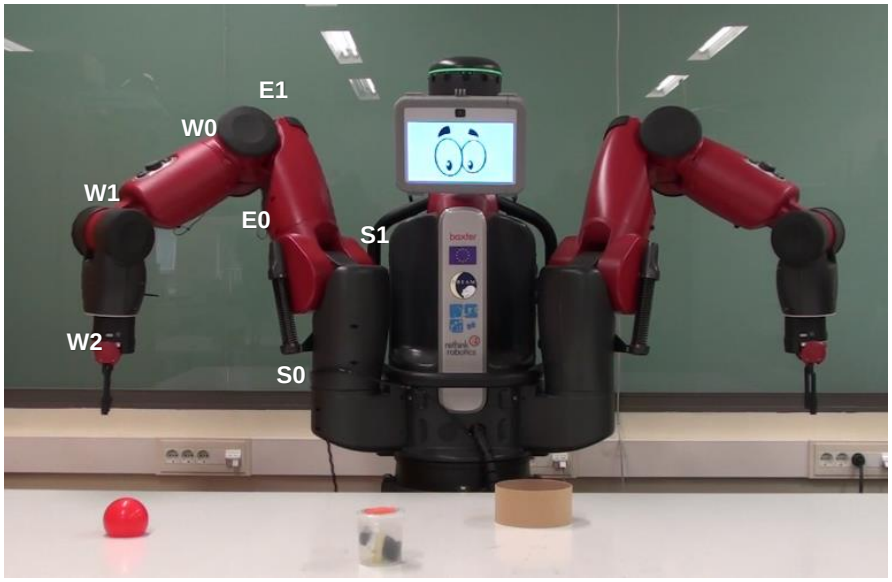

Figura 8-1. Montaje experimental del robot Baxter.

En las siguientes imágenes se puede ver el desarrollo de diferentes acciones en cada instante *t* y *t+1* en el simulador:

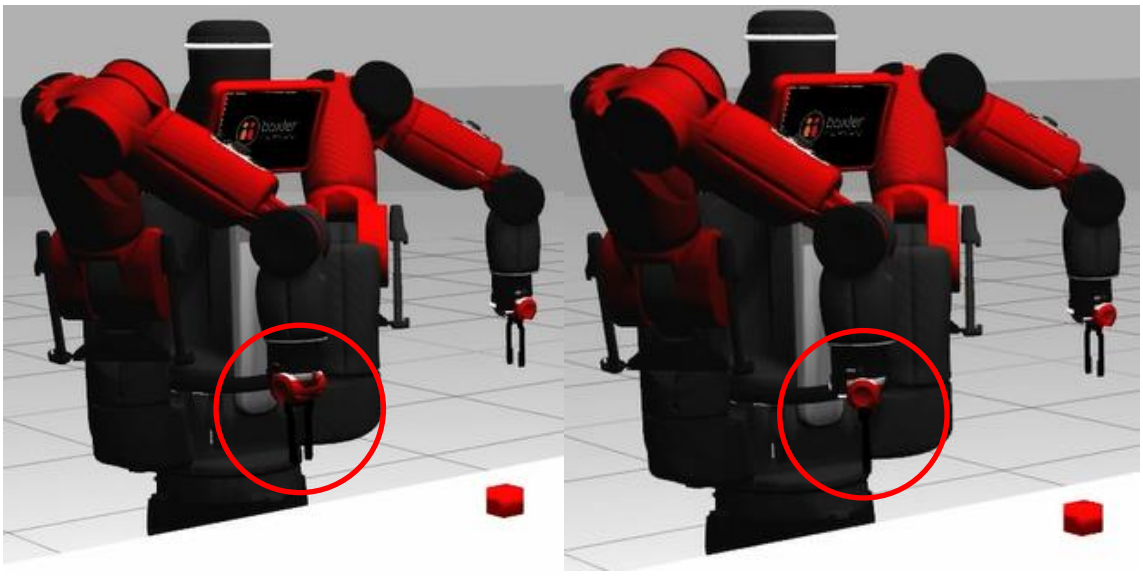

Figura 8-2. Acción en el tiempo t (izquierda) y en t+1 (derecha)

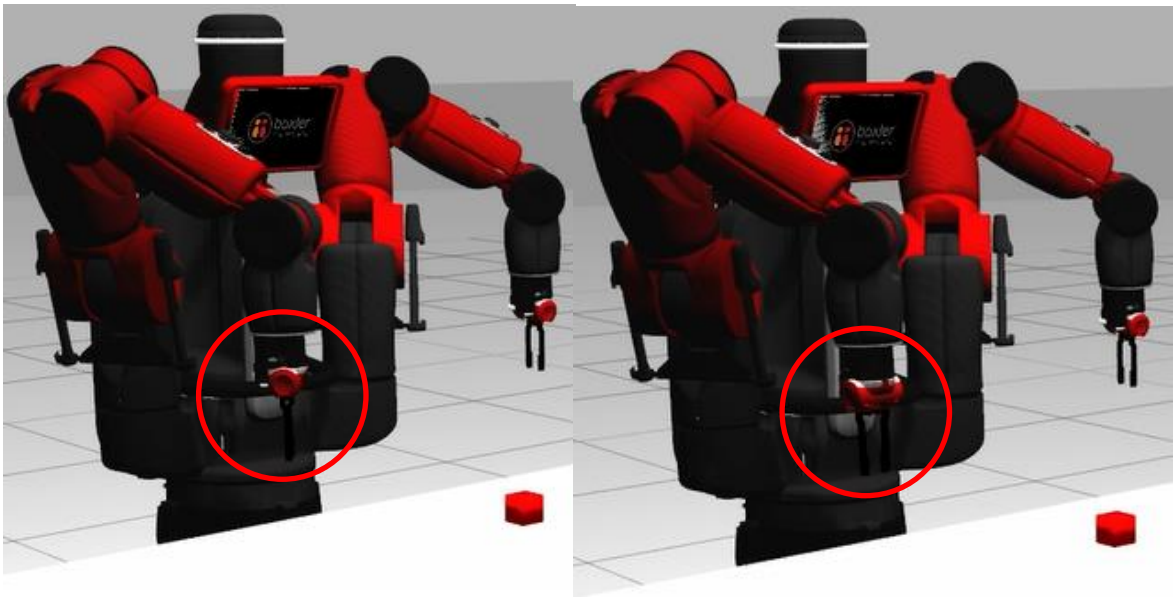

Figura 8-3. Acción en el tiempo t (izquierda) y en t+1 (derecha)

Cada acción es generada por unas entradas y genera unas salidas que son almacenadas en un archivo CSV que se utilizará para entrenar el modelo de red propuesto. El volumen de datos es de 2500 entradas que tal y como se explicará posteriormente se dividirán en conjunto de entrenamiento y de test.

Para la realización del experimento se ha utilizado el algoritmo Adam Optimizer sin modificar ningún parámetro predefinido por Keras, quedando definido como:

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

8.3 Resultados

En este apartado se comentarán los resultados obtenidos en las pruebas anteriormente.

8.3.1 Iteraciones en el aprendizaje.

Partiendo del conjunto de datos anteriores, con el objetivo de fijar un valor para las iteraciones de aprendizaje con la ventana deslizante, se realizaron pruebas de entrenamiento con la función *fit()* y el método *EarlyStopping*. En estas pruebas, se analizaron los siguientes parámetros: tamaño de lote, épocas, tiempo total de entrenamiento, error de test promedio y desviación del error de test. Para esta prueba inicial, se utilizaron diversos tamaños de red, tratando de lograr un error de test inferior al deseado de 5 mm. En este apartado vamos a mostrar los resultados obtenidos con una red de 2 capas ocultas (7-10-10-3).

TAMAÑO DE LOTE	*ÉPOCAS*	*TIEMPO TOTAL*	*ERROR PROMEDIO TEST*	*DESVIACIÓN PROMEDIO*
10	900	211.262335	0.00352655	0.00049449
20	1300	175.469478	0.00333049	0.00043562
50	1700	118.022797	0.00350898	0.00023921
100	2000	47.9984761	0.00375374	0.00025147

Tabla 2. Resultados entrenamiento inicial con función *fit*.

Los resultados obtenidos para cada entrenamiento, mostrados en la anterior tabla muestran que, aunque el tamaño de lote de 10 alcanza resultados muy parecidos al tamaño de lote de 100 y 50 en cuanto al error de test y la desviación del error, el tiempo de cómputo es excesivamente superior. De hecho, la diferencia de tiempo de cómputo para los tamaños de lote de 100 y 50 hace que sea el tamaño de lote de 100 la mejor opción para el entrenamiento. Aunque se ha comentado antes cuando se explicó el algoritmo de descenso de gradiente estocástico, la gran diferencia en el tiempo de cómputo para un tamaño de lote cada vez más pequeño con respecto a uno más grande es debida a que el error se retropropaga hacia todas las neuronas que forman las capas de la red neuronal. Cuanto más grande y complicada sea la red y menor sea el tamaño de lote, mayor será el tiempo de cómputo.

Haciendo uso de la función *EarlyStopping* de Keras, que permite cortar los entrenamientos cuando el error de entrenamiento se estanca o comienza a aumentar, se obtiene la siguiente gráfica para un lanzamiento, observamos en la tabla cómo el número de épocas aumenta al incrementar el tamaño de lote, como era de esperar, aunque no esto no afecta al tiempo de cómputo total ya que hay menos realimentaciones del error. En la Figura 8-2 mostramos la evolución del error de entrenamiento para un caso representativo con mini-lote 100, donde se observa la evolución del error en este caso, y el punto donde el criterio de parada considera que se debe cortar.

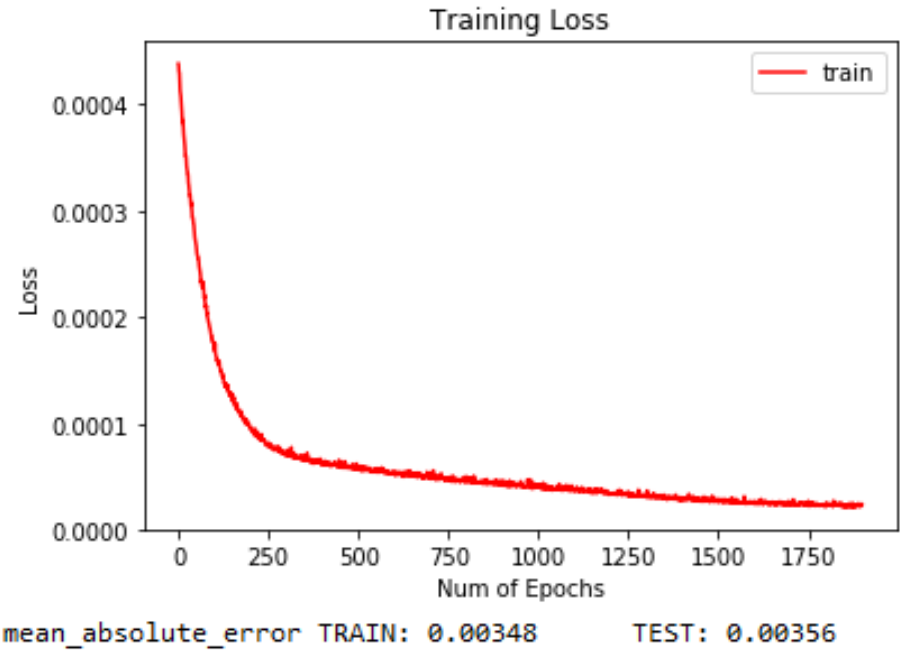

Figura 8-4. Evolución del error de entrenamiento en un entrenamiento *off-line* para un tamaño de *batch* de 100.

Los resultados observables en la tabla 2 muestran que, para un lote o *batch* de 100, las épocas (*epoch*) antes de la estabilización del error rondaban las 2000. Haciendo uso de la siguiente ecuación se comprueba que el número de iteraciones a realizar son.

$$Nº\ iteraciones = epoch * \frac{tamaño_{datos_{entrenamiento}}}{tamaño_{batch}}$$

Para la ejecución de los 45000 pasos, debemos leer un archivo CSV que contiene el conjunto de datos solapados un número n de veces necesario para completar los 45000 pasos, ya que el archivo original solo posee 2500 datos.

Dicho esto, se procedió a la prueba con la ventana deslizante con el objetivo de comprobar si este número de iteraciones era adecuado también utilizando esta aproximación: se seleccionaron 3 redes diferentes, con 3 tamaños de lote cada una.

Tamaños de memoria de la ventana deslizante: 10, 50, 100

Tipologías de red:

- 7 - 10 - 3
- 7 - 10 - 10 - 3
- 7 - 20 - 15 - 10 - 3

Se puede observar en la siguiente tabla resumen los resultados obtenidos. Para la red más pequeña los resultados son bastante malos, con un error medio de 6,5 milímetros, sin embargo, para los dos siguientes tamaños de red los errores de test se encuentran por debajo del límite establecido por el funcionamiento del robot Baxter, es decir, 5 milímetros.

Para la red más compleja se puede ver que los errores son realmente pequeños, pero el tiempo de cómputo es mayor que para una red más sencilla de dos capas ocultas. Por ello se opta por estudiar las gráficas de esta red y no la que da un mejor valor de error de test,

ya que no se considera necesario utilizar una red tan compleja para un problema tan sencillo, viendo también el buen resultado obtenido con dos capas ocultas. Para ver en detalle los resultados de las simulaciones se puede acudir al apartado del anexo 1, tablas 8 y 9.

TIPOLOGÍA DE RED	TAMAÑO DE LOTE	ERROR PROMEDIO	DESVIACIÓN PROMEDIO	TIEMPO PROMEDIO
7X10X3	10	0.01372692	0.00832466	34.62188148
	50	0.00852427	0.00541981	36.55506897
	100	0.00648446	0.00424499	38.61499763
7X10X10X3	10	0.0116705	0.00717024	34.01723059
	50	0.00498963	0.00387032	37.6529328
	100	0.00423824	0.00332971	39.51915081
7X20X15X10X3	10	0.01024839	0.00603171	45.49512832
	50	0.00382238	0.00229704	50.13191764
	100	0.00335692	0.0018752	53.65642198

Tabla 3.Tabla 3. Resumen de resultados con 45000 iteraciones.

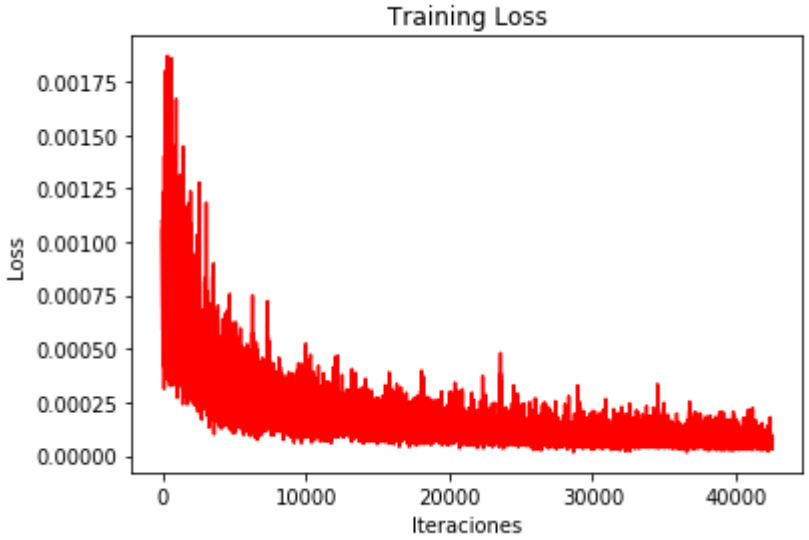

Figura 8-5. Evolución del error de entrenamiento con tamaño de lote 10.

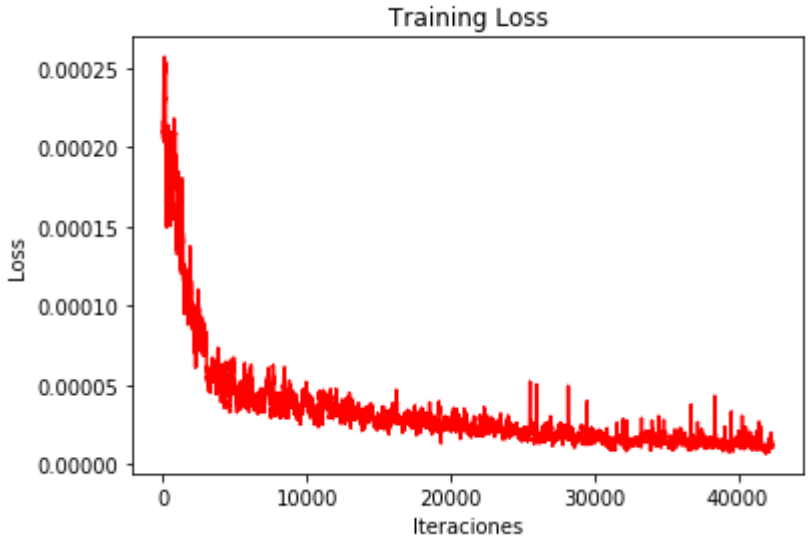

Figura 8-6. Evolución del error de entrenamiento con tamaño de lote 100.

Si se presta atención a las gráficas se puede observar que la gráfica no barre la totalidad de datos hasta acabar en el número 45000. Esto se ha hecho porque, por defecto, la función *train_on_batch()* inicializa los pesos de las redes neuronales con un número aleatorio, que tal y como se ha comprobado, en la mayoría de ocasiones toma un valor demasiado alto. Así en las primeras iteraciones la función de pérdida toma una forma con una pendiente excesivamente alta, con lo cual no se puede apreciar bien la manera en la que decae y se estabiliza la función. Así, los valores de la pérdida que se muestran en las gráficas no tienen en cuenta los primeros 2500 datos de la lista.

Se puede observar que para un tamaño de ventana de 10 las oscilaciones de la función de pérdida (figura 8-5) son mucho mayores que para un tamaño de cien de 100 (figura 8-6), lo cual tiene sentido, ya que cuando se tiende hacia un algoritmo estocástico la gráfica de la función de pérdida tiene mucho más ruido.

Tras lo anteriormente expuesto y analizar las gráficas, se toma la decisión de bajar el número de pasos de entrenamiento para la siguiente prueba a 36000, ya que los 45000 pasos entran la red en exceso y aumenta innecesariamente el tiempo de cómputo, obteniendo la siguiente gráfica de pérdida para una red de 2 capas ocultas de 10 neuronas cada una.

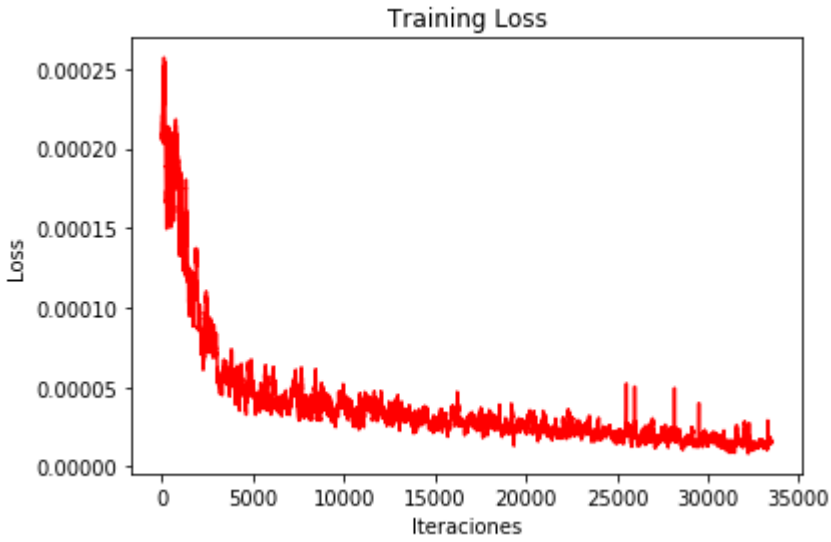

Figura 8-7. Evolución del error de entrenamiento con tamaño de lote 100 con 36000 iteraciones.

Los resultados que se logran alcanzar para una red con dos capas ocultas de 10 neuronas cada una, con 36000 iteraciones en el entrenamiento rondan los 4,2 milímetros (tabla 3), un resultado prácticamente igual al anterior, pero en menos tiempo de cómputo, lo cual favorece un entrenamiento más dinámico.

8.3.2 Tamaños de red y ventana.

En esta prueba se pretende escoger un primer conjunto de tipologías de redes neuronales, así como también el tamaño específico del tamaño de la ventana deslizante para la realización de diferentes pruebas.

El procedimiento es el mismo que el anterior, entrenar el modelo barriendo todo el conjunto de datos con diferentes parámetros de red y memoria con un total de pasos de 36000.

Para llevar a cabo esta prueba se han escogido cuatro posibles tamaños memoria a corto plazo con la ventana deslizante y nueve tipologías de redes diferentes con diferentes capas ocultas y neuronas en cada capa:

Tamaños de memoria de la ventana deslizante: 10, 20, 50, 100

Tipologías de red:

- 7-7-3
- 7-10-3
- 7-7-10-3
- 7-10-10-3
- 7-7-12-8-3
- 7-7-12-10-3
- 7-7-13-10-3
- 7-10-12-8-3
- 7-11-13-10-3

Para cada tipología de red se entrenará el modelo una única vez usando los distintos tamaños de memoria de la ventana deslizante. Los resultados en los errores de test obtenidos después de entrenar el modelo de todas las simulaciones se adjuntan en el anexo 1, tablas 10 y 11.

Como se ha comentado previamente el error de test máximo que se pretende cometer es de cinco milímetros. Con todo lo comentado y analizando la tabla del anexo 1 se pueden comprobar dos cosas:

- Los tamaños de ventana pequeños, menores a 50 no dan por lo general errores de test bajos comparados con unos tamaños más grandes, así que se toma la decisión de realizar las pruebas posteriores con una ventana de tamaños de 50 y 100.
- Las redes de un mayor número de capas ocultas son las que mejor resultados dan, aunque el tiempo de cómputo es ligeramente mayor. Dentro de estas, las redes con una segunda capa oculta de 10 neuronas dan un mejor resultado que las de 7. Es importante comentar esto, ya que, en ciertas ocasiones, para la resolución de diversos problemas, el utilizar una capa oculta con un menor número de neuronas permite hacer una simplificación del problema, lo cual posibilita un aprendizaje mejor y más rápido.

Tras las conclusiones anteriores, se pasa ahora a realizar pruebas más exhaustivas con los tamaños de red y ventana seleccionados.

De nuevo, todos los resultados de las simulaciones están contenidos en el anexo 1, tablas 12 y 13. Sin embargo, se muestra a continuación una pequeña tabla resumen.

TIPOLOGÍA	TAMAÑO DE LOTE	ERROR DE TEST PROMEDIO	DESVIACIÓN PROMEDIO	TIEMPO ITERACIÓN
7X10X12X8X3	100	0.0037964	0.002747789	0.00085502
7X10X10X3	100	0.00427959	0.003020119	0.00074558
7X11X13X10X3	100	0.00442606	0.003315504	0.0008285
7X10X12X8X3	50	0.00541729	0.00362284	0.00077894
7X11X13X10X3	50	0.00548326	0.004048571	0.00078701
7X7X12X10X3	100	0.00579642	0.004043369	0.00083235
7X10X10X3	50	0.00617647	0.004045976	0.00068202
7X7X12X10X3	50	0.00634417	0.004660283	0.0007568
7X10X3	100	0.00657444	0.004587475	0.00061664
7X7X13X10X3	100	0.00663669	0.004744039	0.00081437
7X7X13X10X3	50	0.00673467	0.004675288	0.00081544
7X10X3	50	0.00797636	0.00540884	0.00066582

Tabla 4. Resumen tipologías de red y tamaño de la ventana deslizante.

En esta tabla resumen aparecen todas las combinaciones con los resultados promediados de las 10 simulaciones. Se puede ver claramente que la tipología de red 7-10-12-8-3 ha alcanzado unos resultados bastante buenos, con un error mucho menor que el máximo error que puede cometer el *gripper*. Sin embargo, se trata de una red muy complicada para un problema sencillo como el que se plantea. Aparte, al ser más compleja la red, los tiempos de iteración son mayores, y para un aprendizaje online interesa un entrenamiento más dinámico. Dicho esto, se descarta para continuar realizando pruebas con ella.

Se opta por lo tanto por la segunda combinación de la tabla: una red de dos capas ocultas de diez neuronas cada una de ellas, con un tamaño de ventana de 100 datos y con un error promedio asumible, ya que sigue por debajo del máximo error permitido. De hecho, si se compara el error de la segunda fila con el de la tercera, se puede observar que los resultados son muy semejantes, pero la complejidad de la red neuronal de la tercera fila valida la opción de usar el modelo de dos capas ocultas.

Las otras combinaciones son desechables, bien porque son redes más complejas o bien porque los resultados finales superan un error de 5 milímetros o ambas.

Para hacerse una idea de la precisión de las predicciones que es capaz de realizar el modelo seleccionado (7x10x10x3) se muestra a continuación las salidas originales, contenidas en el archivo de datos proporcionado por el GII, de color azul y las predicciones realizadas por el modelo para cada las tres componentes del espacio x, y, z, de color naranja:

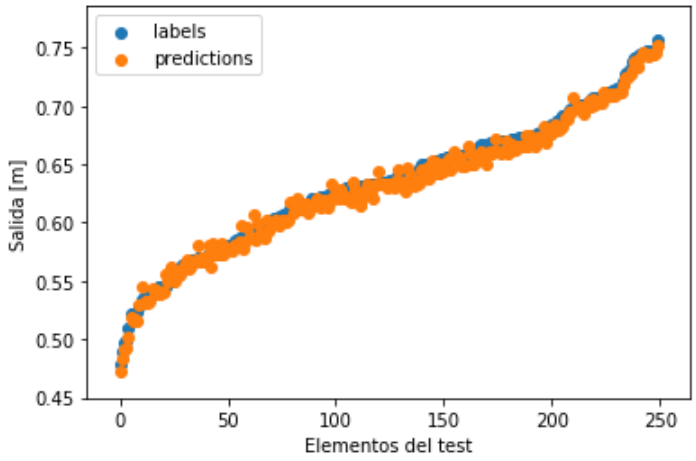

Figura 8-8. Gráfica de etiquetas VS predicciones en x.

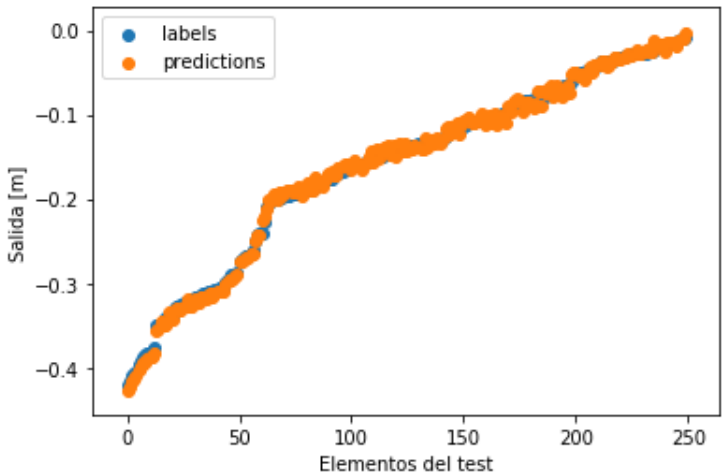

Figura 8-9. Gráfica de etiquetas VS predicciones en y.

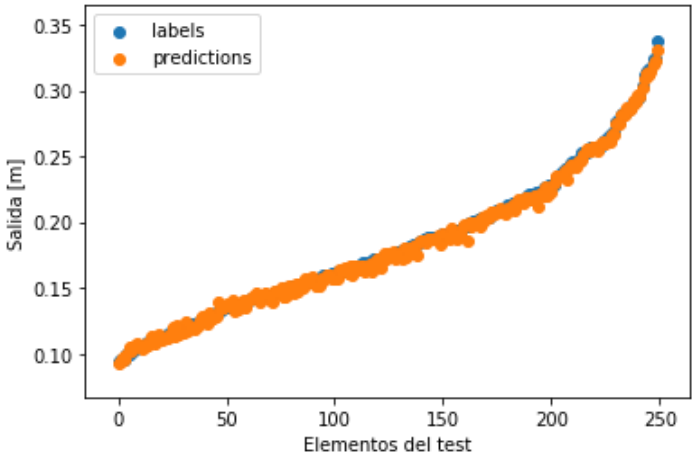

Figura 8-10. Gráfica de etiquetas VS predicciones en z.

Los resultados obtenidos con el modelo, tal y como se pueden ver en las anteriores gráficas son muy satisfactorios. El error del conjunto de test promedio en *x*, de casi 5 milímetros, es el más alto de todos. En el eje *y* el error se reduce a 4 milímetros y finalmente, en el eje *z*, el error supera, los 3 milímetros sin llegar a 4. De hecho, excepto en algunos puntos de las gráficas, donde se puede ver mínimos tramos de color azul representando los valores de las etiquetas, en cada gráfica las predicciones se ajustan perfectamente al valor de salida.

8.3.3 Pasos de entrenamiento

En esta prueba, se van a utilizar todos los parámetros fijados con anterioridad: tamaño de red de 7x10x10x3, un tamaño de ventana de 100, y un número de pasos variable. Para garantizar un número de pasos similares para cada repetición se dividirá el número de pasos iniciales, 36000, entre un factor que será igual al número de repeticiones seleccionadas.

Los resultados del entrenamiento obtenidos para esta configuración se pueden ver en la siguiente tabla.

NÚMERO PASOS	ERROR PROMEDIO	DESVIACIÓN PROMEDIO	TIEMPO TOTAL
N=1	0.004279595	0.00302012	26.7656534
N=2	0.004168559	0.00303472	24.5478911
N=5	0.00467748	0.00346552	24.6014113
N=10	0.004819215	0.00342163	25.5162106
N=100	0.004697487	0.00383523	25.0855944

Tabla 5. Resultados de entrenamiento con pasos de entrenamiento (1).

Para cada posible opción se la lanzado una vez el programa. Se ve que los resultados solamente mejoran con dos repeticiones el conjunto de datos. Aunque los datos son bastante similares, un aumento en el error se puede achacar a que, al aumentar el número de repeticiones, el modelo aprende en exceso el conjunto de datos, es decir, se sobreentrena el modelo. Así, cuando se produce un cambio en el conjunto de datos, al modelo le cuesta más adecuarse y comete un error más grande.

Para validar los resultados anteriores se vuelven a lanzar diez ensayos, pero solo para las opciones de n=2, n=5 y n=10. La tabla completa de resultados se puede encontrar en el anexo 1, tablas 14 y 15, aunque a continuación se muestra una tabla resumen.

NÚMERO DE REPETICIONES	ERROR PROMEDIO	DESVIACIÓN PROMEDIO
N=1	0.00427959	0.00302012
N=2	0.00406376	0.00301447
N=5	0.00447889	0.00320839
N=10	0.00486568	0.00336788

Tabla 6. Resultados de entrenamiento con pasos de entrenamiento (2).

Efectivamente, los resultados solo mejoran para n=2. En las siguientes gráficas podemos ver las predicciones del modelo frente a las etiquetas de los datos del conjunto de test obtenidos del robot real, en este caso n=2. Aunque en la tabla la diferencia entre los resultados de las dos primeras filas es fácilmente apreciable, en las gráficas no se puede observar de manera tan sencilla un desfase de dos décimas de milímetro. Aun así, se puede ver que los resultados siguen siendo realmente buenos.

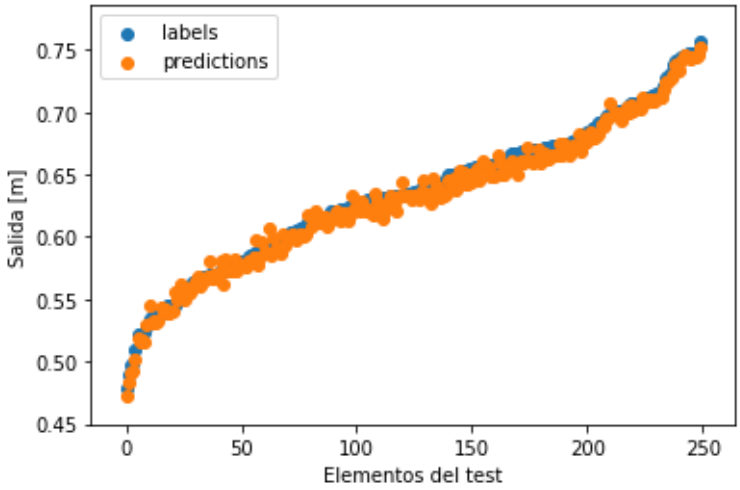

Figura 8-11. Gráfica de Etiquetas VS predicciones en x.

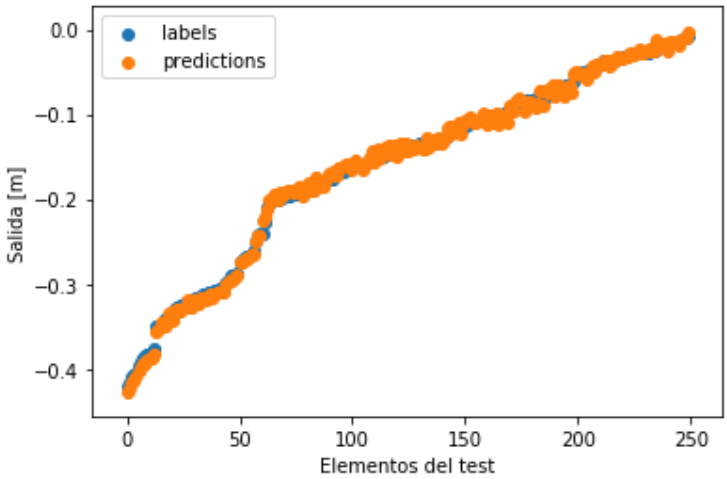

Figura 8-12. Gráfica de Etiquetas VS predicciones en y.

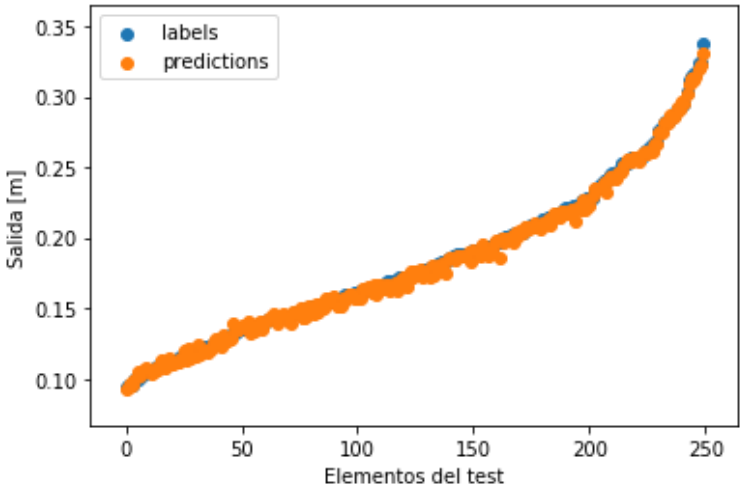

Figura 8-13. Gráfica de Etiquetas VS predicciones en z.

Los resultados obtenidos con el modelo que se pueden ver en las anteriores gráficas, tal y como se esperaba tras el análisis de los resultados de la tabla, son muy similares a los anteriormente mostrados en las figuras 7-13, 7-14 y 7-15. El error de test en x, sigue siendo prácticamente de cinco milímetros. En el eje y el error se reduce a 3.8 milímetros y finalmente, en el eje z, disminuye el error, que llega a bajar mínimamente de los tres milímetros. De hecho, en la figura 7-19 se puede ver que los datos de las predicciones prácticamente se sobreponen a los datos de la salida proporcionado por el del conjunto de test.

8.3.4 Nueva gestión de memoria a corto plazo

Manteniendo la tipología de la red (7x10x10x3), el tamaño de ventana (100), el número de iteraciones (17949) y el número de pasos de la prueba anterior (n=2), se proponen los siguientes tamaños de mini-lotes (el tamaño de uno se corresponde con el funcionamiento de un algoritmo estocástico):

- Tamaño = 1
- Tamaño = 10
- Tamaño = 50

De nuevo, para los tamaños de 10 y 50 se realizaron 10 pruebas. Sin embargo, debido a que el tiempo de cómputo correspondiente al algoritmo estocástico era excesivamente largo y una simulación tardaba alrededor de 3000 segundos, se realizaron 5 pruebas únicamente.

En el apartado de anexo 1, tablas 16 y 17 se puede ver la tabla completa de los resultados de las 25 simulaciones. La siguiente tabla muestra los promedios de las pruebas realizadas:

TAMAÑO MINIBATCH	ERROR PROMEDIO	DESVIACIÓN PROMEDIO	TIEMPO PROMEDIO
1	0.00609307	0.00445926	2957.54515
10	0.00421922	0.00289848	396.52262
50	0.00430667	0.00292233	76.8443003

Tabla 7. Resumen resultados de la nueva gestión de la memoria

Como se puede observar en la tabla 6, los errores son muy similares a los del subapartado anterior, recopilados en la tabla 5, para ambos tamaños de mini-lote, pero no

es posible bajar el nivel de error de test. Evidentemente, los tiempos de cómputo, aumentan con respecto a tiempos los obtenidos con la ventana deslizante usando repeticiones en el entrenamiento.

También, como se puede ver en la anterior gráfica, los resultados obtenidos con los tamaños de 10 y de 50 son muy similares, pero el tiempo de cómputo para el tamaño de 10 es 5 veces mayor que el necesario para completar un entrenamiento con un tamaño de 50.

A continuación, se muestran las gráficas de las predicciones con sus correspondientes etiquetas para un entrenamiento con un tamaño de mini-lote de 50. El error en *x* sería de 3,9 milímetros, en *y*, 4,7 milímetros y en *z,* 3,6 milímetros.

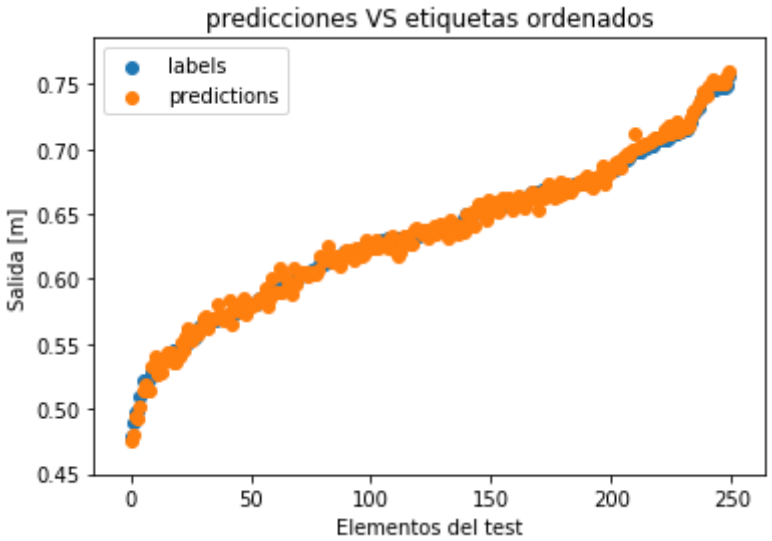

Figura 8-14. Gráfica de Etiquetas VS predicciones en x.

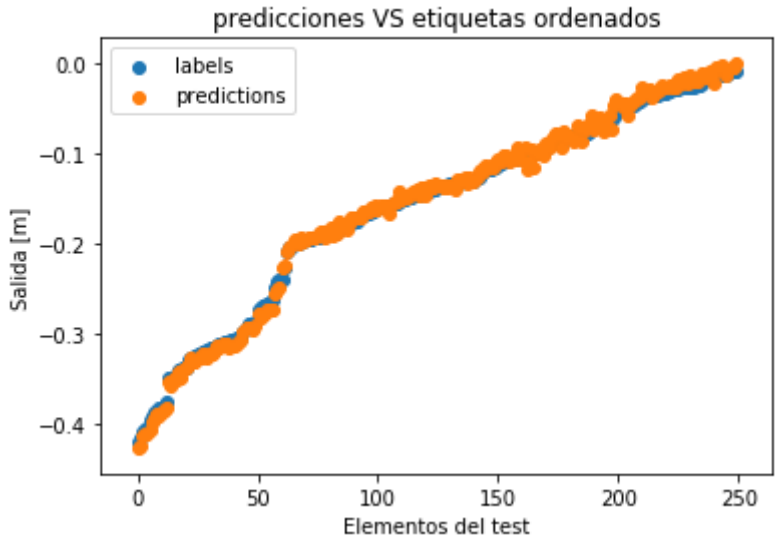

Figura 8-15. Gráfica de Etiquetas VS predicciones en y.

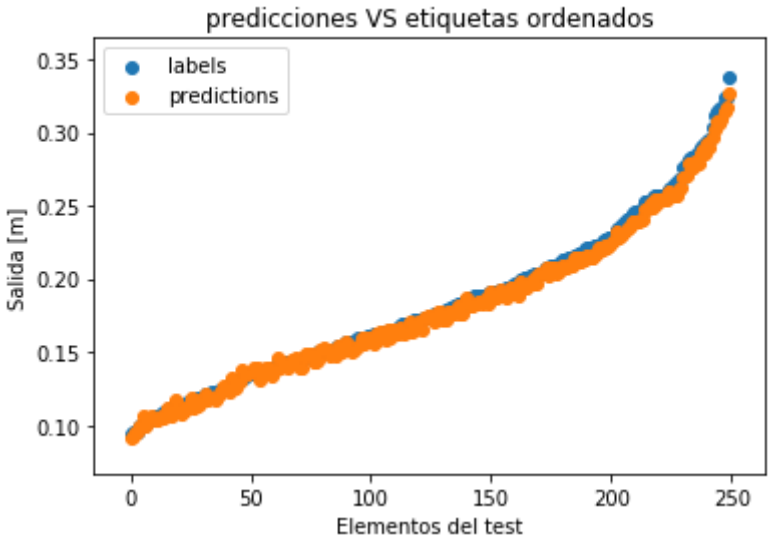

Figura 8-16. Gráfica de Etiquetas VS predicciones en z.

Es interesante comparar las predicciones de este nuevo sistema de gestión de memoria con los resultados obtenidos con un entrenamiento *off-line* usando mini-lotes como el utilizado en el apartado 8.3.1, ya que en ambos casos se utiliza la función *fit* para realizar el entrenamiento.

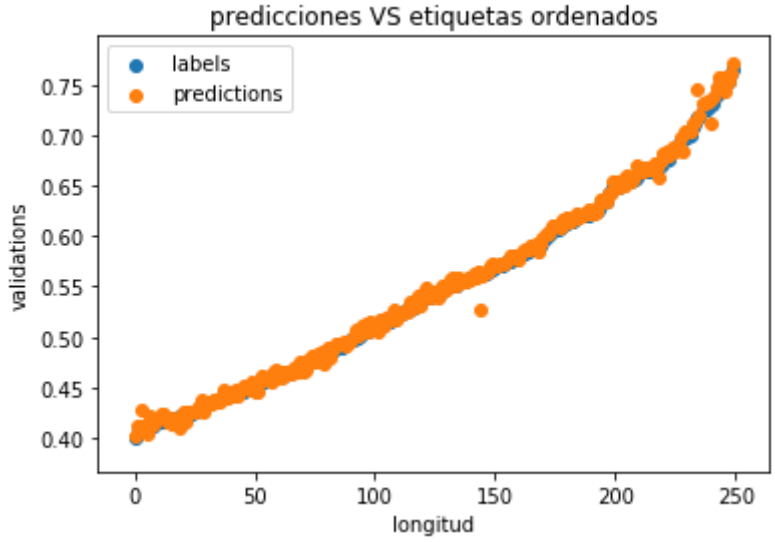

Figura 8-17. Gráfica de Etiquetas VS predicciones en x.

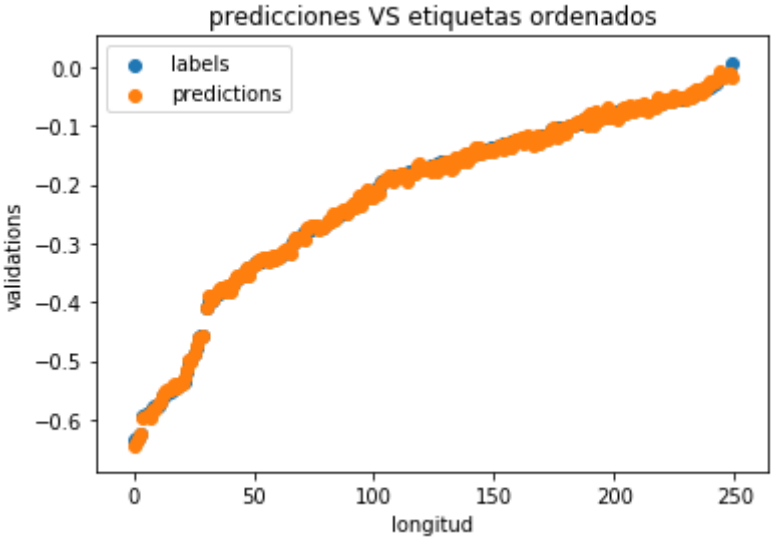

Figura 8-18. Gráfica de Etiquetas VS predicciones en y.

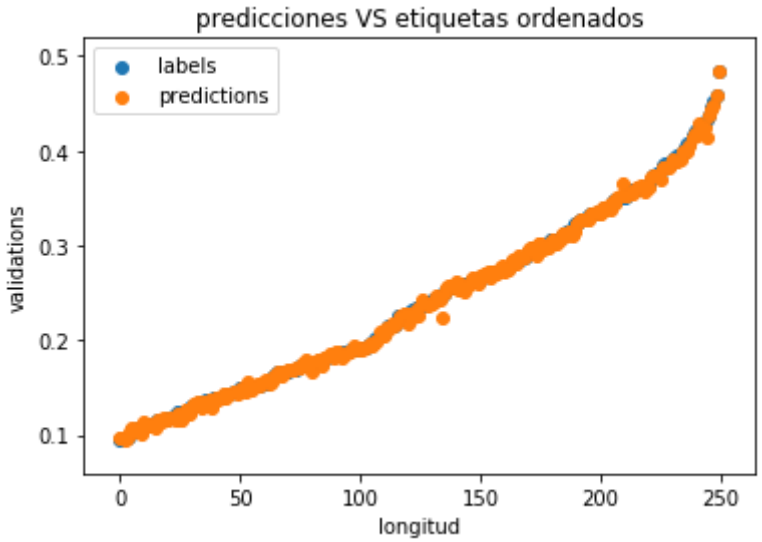

Figura 8-19. Gráfica de Etiquetas VS predicciones en z.

Debido a que los datos de test no son los mismos, las gráficas no tienen exactamente la misma forma, pero lo que sí se puede comprobar es que, sobre todo en el eje *x*, los ajustes son más precisos y forman una curva más superpuesta a los datos azules (salvo en un dato) en el segundo conjunto de gráficas. Igualmente pasa en el eje *z*. En el eje *y* sí que se puede comprobar que las predicciones se ajustan mejor a las etiquetas en el segundo conjunto de gráficas.

9 CONCLUSIONES

Se ha propuesto una estrategia de aprendizaje on-line de modelos de mundo basada en el uso de técnicas de *Deep Learning*, y también se ha planteado una nueva gestión de la memoria a corto plazo. Una vez obtenida la combinación se ha probado un nuevo sistema de gestión de memoria y se ha comparado con un entrenamiento con la función *fit*, típica de un entrenamiento completamente off-line.

A lo largo de todo el trabajo se van alcanzando los subobjetivos planteados:

- Analizar el proceso de aprendizaje actual en la arquitectura MDB.
- Analizar los algoritmos de aprendizaje *on-line* para redes neuronales dentro del campo de *Deep Learning*.
- Desarrollar un procedimiento de aprendizaje on-line teniendo en cuenta las limitaciones en el tiempo de cómputo.
- Realizar un estudio comparativo formal de las diferentes opciones para el aprendizaje.
- Proponer un procedimiento de gestión de la memoria a corto plazo y una parametrización del algoritmo de aprendizaje seleccionado.

Sobre un problema real de robótica cognitiva proporcionado por el GII, se ha realizado un análisis de la parametrización óptima del sistema propuesto, para su aplicación en tiempo real.

Se ha podido concluir que, con un tamaño de ventana de 100, una red de 2 capas ocultas de 10 neuronas, y el uso del algoritmo ADAM, se pueden realizar entrenamientos que proporcionan muy buenos resultados con un tiempo de cómputo compatible con un aprendizaje on-line.

El nuevo sistema de gestión de memoria propuesto da buenos resultados, pero no logra alcanzar un error de test tan bajo como con el entrenamiento inicial con la función *fit* ni tampoco con el entrenamiento usando la ventana deslizante con dos pasos por iteración en el entrenamiento.

En todas las pruebas realizadas en este Trabajo de Fin de Grado se utilizan el mismo conjunto de datos, con lo cual sería beneficioso probar en un futuro estas mismas pruebas para un conjunto de datos diferente y comprobar la validez del modelo. Otra línea de trabajo futuro se puede centrar en el estudio de otros parámetros como puede ser el ratio de aprendizaje del algoritmo de optimización del *Adam Optimizer* o la utilización de otro algoritmo.

10 ANEXOS

10.1 Anexo 1: Tablas de resultados

TIPOLOGÍA	N	PRUEBA	ERROR X	ERROR Y	ERROR Z	ERROR PROMEDIO
7X10X3	10	1	0.01057368	0.009134145	0.01611949	0.01194244
7X10X3	10	2	0.02142855	0.013987734	0.00941098	0.01494242
7X10X3	10	3	0.01857924	0.010929038	0.0133794	0.01429589
7X10X3	50	1	0.00558157	0.010663421	0.00864196	0.00829565
7X10X3	50	2	0.01279645	0.006977276	0.01294643	0.01090672
7X10X3	50	3	0.00556081	0.005712662	0.00783789	0.00637046
7X10X3	100	1	0.00512897	0.005944654	0.00478622	0.00528662
7X10X3	100	2	0.00852632	0.00499731	0.00655514	0.00669293
7X10X3	100	3	0.00632147	0.003971078	0.01122898	0.00717384
7X10X10X3	10	1	0.01649631	0.015505496	0.00808312	0.01336164
7X10X10X3	10	2	0.01411011	0.010312726	0.01051662	0.01164649
7X10X10X3	10	3	0.00781695	0.006952772	0.01524038	0.01000337
7X10X10X3	50	1	0.00479807	0.00532449	0.00242219	0.00418158
7X10X10X3	50	2	0.00649816	0.005821553	0.00444609	0.0055886
7X10X10X3	50	3	0.00473884	0.005481769	0.00537556	0.00519872
7X10X10X3	100	1	0.00499787	0.005058761	0.00322399	0.00442687
7X10X10X3	100	2	0.00349113	0.005303367	0.00388367	0.00422606
7X10X10X3	100	3	0.00337948	0.004203092	0.00460284	0.0040618
7X20X15X10X3	10	1	0.01200896	0.005970855	0.011829	0.00993627
7X20X15X10X3	10	2	0.01162896	0.005685255	0.01163255	0.00964892
7X20X15X10X3	10	3	0.01630256	0.009941167	0.00723619	0.01115997
7X20X15X10X3	50	1	0.00344119	0.003210254	0.00315602	0.00326915
7X20X15X10X3	50	2	0.00352006	0.004210611	0.00408131	0.00393733
7X20X15X10X3	50	3	0.00474006	0.003960611	0.00408131	0.00426066
7X20X15X10X3	100	1	0.00204756	0.003039217	0.00392869	0.00300515
7X20X15X10X3	100	2	0.006187	0.002749052	0.00281544	0.00391716
7X20X15X10X3	100	3	0.0038634	0.002800423	0.00278152	0.00314845

Tabla 8. Resumen de resultados con 45000 iteraciones(I)

TIPOLOGÍA	DESV. X	DESV. Y	DESV. Z	DESVIACIÓN PROM	TIEMPO ITERACION	TIEMPO TOTAL
7X10X3	0.00727	0.00717	0.00720	0.0072	0.00077007	34.64
7X10X3	0.01084	0.00848	0.00568	0.0083	0.00081558	36.69
7X10X3	0.01212	0.00845	0.00771	0.0094	0.00087863	32.53
7X10X3	0.00368	0.00815	0.00456	0.0055	0.00085589	38.47
7X10X3	0.00804	0.00535	0.00444	0.0059	0.00086275	33.78
7X10X3	0.00405	0.00537	0.00514	0.0049	0.00083237	37.41
7X10X3	0.00387	0.00499	0.00308	0.0040	0.00096068	38.13
7X10X3	0.00276	0.00301	0.00416	0.0033	0.00092799	36.67
7X10X3	0.00498	0.00365	0.00769	0.0054	0.00091418	41.05
7X10X10X3	0.00882	0.00875	0.00566	0.0077	0.00072624	32.67
7X10X10X3	0.00785	0.00581	0.00676	0.0068	0.00074129	33.35
7X10X10X3	0.00598	0.00515	0.00975	0.0070	0.00080084	36.03
7X10X10X3	0.00383	0.00505	0.00185	0.0036	0.00076522	34.40
7X10X10X3	0.00568	0.00450	0.00291	0.0044	0.0007568	34.02
7X10X10X3	0.00312	0.00343	0.00445	0.0037	0.0007908	35.55
7X10X10X3	0.00396	0.00358	0.00225	0.0033	0.00079909	35.88
7X10X10X3	0.00494	0.00234	0.00265	0.0033	0.00081706	36.68
7X10X10X3	0.00487	0.00357	0.00181	0.0034	0.00082394	36.99
7X20X15X10X3	0.00566	0.00446	0.00767	0.0059	0.00102704	46.21
7X20X15X10X3	0.00569	0.00496	0.00668	0.0058	0.00992704	45.31
7X20X15X10X3	0.00751	0.00581	0.00584	0.0064	0.00091069	44.97
7X20X15X10X3	0.00201	0.00209	0.00152	0.0019	0.00107013	55.10
7X20X15X10X3	0.00175	0.00265	0.00313	0.0025	0.00102666	49.15
7X20X15X10X3	0.00175	0.00265	0.00313	0.0025	0.00102666	46.15
7X20X15X10X3	0.00165	0.00241	0.00143	0.0018	0.000988	54.36
7X20X15X10X3	0.00290	0.00173	0.00124	0.0020	0.0009959	54.72
7X20X15X10X3	0.00220	0.00193	0.00138	0.0018	0.00104443	51.89

Tabla 9. Resumen de resultados con 45000 iteraciones (II)

TIPOLOGÍA RED	*LOTE*	ERROR X	ERROR Y	ERROR Z	ERROR PROMEDIO
7X7X13X10X3	100	0.00262117	0.0035252	0.00562572	0.0039240
7X7X12X10X3	100	0.00482455	0.00631666	0.00406012	0.0050671
7X11X13X10X3	100	0.00332486	0.0046337	0.00802751	0.0053287
7X10X12X8X3	100	0.00341065	0.01031806	0.00316623	0.0056316
7X10X12X8X3	20	0.00695119	0.00618345	0.00422193	0.0057855
7X10X3	100	0.00743184	0.00534501	0.00462386	0.0058002
7X10X12X8X3	50	0.00475491	0.00640935	0.00673085	0.0059650
7X7X13X10X3	50	0.00426004	0.00526415	0.00855158	0.0060253
7X11X13X10X3	50	0.00484061	0.00452472	0.00982415	0.0063965
7X7X12X10X3	50	0.0040739	0.00999758	0.00836042	0.0074773
7X10X10X3	50	0.00826991	0.00583248	0.00979415	0.0079655
7X10X10X3	100	0.01014113	0.00548096	0.01112824	0.0089168
7X11X13X10X3	20	0.00865677	0.00419512	0.01690582	0.0099192
7X7X12X8X3	100	0.01081549	0.00796125	0.01178671	0.0101878
7X10X3	50	0.0133696	0.00765106	0.01200272	0.0110078
7X7X10X3	100	0.00946246	0.00966024	0.01466457	0.0112624
7X10X10X3	20	0.01243563	0.00574208	0.01595847	0.0113787
7X7X12X8X3	50	0.01302128	0.0075421	0.01359872	0.0113874
7X10X12X8X3	10	0.00764869	0.00800218	0.01925242	0.0116344
7X7X13X10X3	10	0.01540709	0.00755799	0.01364597	0.0122037
7X7X10X3	50	0.01379182	0.00909298	0.01411034	0.0123317
7X7X12X10X3	20	0.01607075	0.00548074	0.01545163	0.0123344
7X7X12X8X3	20	0.01605025	0.00732967	0.01365257	0.0123442
7X7X13X10X3	20	0.01564991	0.0067921	0.01505653	0.0124995
7X10X3	10	0.02153841	0.00585972	0.01119174	0.0128633
7X7X12X10X3	10	0.01725619	0.00883796	0.0141517	0.0134153
7X7X3	100	0.01649377	0.00906422	0.01693527	0.0141644
7X7X10X3	20	0.01818453	0.00640864	0.01877009	0.0144544
7X10X3	20	0.02423351	0.00613203	0.01509168	0.0151524
7X7X3	50	0.02425694	0.00971955	0.01336521	0.0157806
7X10X10X3	10	0.02226542	0.00801471	0.01731645	0.0158655
7X7X3	20	0.021845	0.01341771	0.01275181	0.0160048
7X11X13X10X3	10	0.01826431	0.00768534	0.02241599	0.0161219
7X7X10X3	10	0.01684995	0.01221457	0.02018031	0.0164149
7X7X12X8X3	10	0.01526913	0.01089711	0.02611339	0.0174265
7X7X3	10	0.02414879	0.01959224	0.00966475	0.0178019

Tabla 10. Preselección de tipologías y tamaño de ventana (I).

TIPOLOGÍA RED	LOTE	DESV. X	DESV. Y	DESV. Z	DESV. PROM.	TIEMPO ITERACIÓN	TIEMPO TOTAL
7X7X13X10X3	100	0.0020	0.0027	0.0033	0.0027	0.00227	81.87
7X7X12X10X3	100	0.0035	0.0047	0.0033	0.0039	0.00211	75.84
7X11X13X10X3	100	0.0026	0.0030	0.0068	0.0042	0.00293	105.53
7X10X12X8X3	100	0.0028	0.0049	0.0023	0.0033	0.00246	88.59
7X10X12X8X3	20	0.0052	0.0043	0.0026	0.0040	0.00234	84.27
7X10X3	100	0.0053	0.0041	0.0029	0.0041	0.00154	55.59
7X10X12X8X3	50	0.0038	0.0046	0.0060	0.0048	0.00235	84.46
7X7X13X10X3	50	0.0032	0.0040	0.0056	0.0043	0.00229	82.45
7X11X13X10X3	50	0.0041	0.0028	0.0081	0.0050	0.00281	101.14
7X7X12X10X3	50	0.0026	0.0068	0.0053	0.0049	0.00207	74.39
7X10X10X3	50	0.0060	0.0038	0.0070	0.0056	0.00180	64.78
7X10X10X3	100	0.0064	0.0040	0.0075	0.0060	0.00196	70.61
7X11X13X10X3	20	0.0051	0.0030	0.0114	0.0065	0.00268	96.57
7X7X12X8X3	100	0.0061	0.0057	0.0070	0.0063	0.00232	83.54
7X10X3	50	0.0080	0.0048	0.0044	0.0057	0.00153	54.91
7X7X10X3	100	0.0054	0.0064	0.0088	0.0069	0.00185	66.76
7X10X10X3	20	0.0071	0.0039	0.0098	0.0069	0.00176	63.17
7X7X12X8X3	50	0.0063	0.0075	0.0084	0.0074	0.00225	80.99
7X10X12X8X3	10	0.0055	0.0055	0.0073	0.0061	0.00230	82.66
7X7X13X10X3	10	0.0066	0.0058	0.0097	0.0074	0.00208	74.95
7X7X10X3	50	0.0075	0.0061	0.0094	0.0077	0.00171	61.42
7X7X12X10X3	20	0.0076	0.0038	0.0096	0.0070	0.00198	71.16
7X7X12X8X3	20	0.0087	0.0049	0.0091	0.0076	0.00225	81.06
7X7X13X10X3	20	0.0083	0.0049	0.0102	0.0078	0.00209	75.05
7X10X3	10	0.0105	0.0050	0.0065	0.0074	0.00142	51.20
7X7X12X10X3	10	0.0068	0.0056	0.0105	0.0076	0.00195	70.06
7X7X3	100	0.0084	0.0079	0.0107	0.0090	0.00158	57.01
7X7X10X3	20	0.0086	0.0056	0.0133	0.0092	0.00175	62.98
7X10X3	20	0.0090	0.0039	0.0068	0.0066	0.00155	55.71
7X7X3	50	0.0113	0.0077	0.0074	0.0088	0.00156	56.17
7X10X10X3	10	0.0098	0.0054	0.0117	0.0089	0.00173	62.24
7X7X3	20	0.0130	0.0083	0.0070	0.0094	0.00147	52.85
7X11X13X10X3	10	0.0079	0.0048	0.0166	0.0098	0.00264	95.11
7X7X10X3	10	0.0081	0.0057	0.0136	0.0091	0.00168	60.57
7X7X12X8X3	10	0.0078	0.0069	0.0161	0.0102	0.00218	78.57
7X7X3	10	0.0182	0.0129	0.0066	0.0126	0.00143	51.50

Tabla 11. Preselección de tipologías y tamaño de ventana (II)

TIPOLOGÍA RED	*LOTE*	Nº PRUEBA	ERROR X	ERROR Y	ERROR Z	ERROR PROMEDIO
7X10X3	50	1	0.00718714	0.00911179	0.00751647	0.00793847
7X10X3	50	2	0.0089888	0.01165266	0.00581324	0.00881823
7X10X3	50	3	0.00768357	0.00533699	0.01155525	0.00819194
7X10X3	50	4	0.00593118	0.01076798	0.0087171	0.00847209
7X10X3	50	5	0.00624537	0.00680544	0.00671015	0.00658699
7X10X3	50	6	0.006701	0.00829187	0.00999656	0.00832981
7X10X3	50	7	0.01088651	0.0089039	0.00249585	0.00742875
7X10X3	50	8	0.00998311	0.00662194	0.0092446	0.00861655
7X10X3	50	9	0.00662055	0.00804441	0.00737514	0.0073467
7X10X3	50	10	0.00760189	0.00669894	0.00980148	0.00803411
7X10X3	100	1	0.00464317	0.0076428	0.0052698	0.00585192
7X10X3	100	2	0.00551506	0.0060574	0.00772343	0.00643196
7X10X3	100	3	0.00723401	0.00764313	0.00829824	0.00772513
7X10X3	100	4	0.00824125	0.0098157	0.0068159	0.00829095
7X10X3	100	5	0.00387383	0.00390915	0.0069445	0.00490916
7X10X3	100	6	0.00734877	0.0090681	0.00457331	0.00699673
7X10X3	100	7	0.00564994	0.00681033	0.00604346	0.00616791
7X10X3	100	8	0.00464801	0.00457684	0.00815016	0.00579167
7X10X3	100	9	0.00557983	0.00860103	0.00513678	0.00643921
7X10X3	100	10	0.00715044	0.00612183	0.008147	0.00713976
7X10X10X3	50	1	0.00537565	0.00531899	0.00456819	0.00508761
7X10X10X3	50	2	0.00795251	0.00509363	0.00268086	0.00524234
7X10X10X3	50	3	0.0053911	0.00842569	0.00747508	0.00709729
7X10X10X3	50	4	0.00565925	0.00545918	0.00535665	0.0054917
7X10X10X3	50	5	0.00482584	0.0059995	0.01087794	0.00723443
7X10X10X3	50	6	0.00932371	0.00645201	0.00537487	0.0070502
7X10X10X3	50	7	0.00459484	0.00757164	0.01142289	0.00786312
7X10X10X3	50	8	0.00600844	0.01177035	0.00611169	0.00796349
7X10X10X3	50	9	0.00472921	0.00601318	0.00235946	0.00436728
7X10X10X3	50	10	0.00472918	0.00601328	0.00235942	0.00436729
7X10X10X3	100	1	0.00513987	0.00328357	0.00509296	0.00450547
7X10X10X3	100	2	0.00355753	0.00267082	0.00426277	0.00349704
7X10X10X3	100	3	0.00392658	0.00419454	0.00208245	0.00340119
7X10X10X3	100	4	0.00578714	0.00341574	0.0045408	0.00458122
7X10X10X3	100	5	0.0041784	0.00312126	0.00325132	0.00351699
7X10X10X3	100	6	0.00432396	0.00418405	0.00596903	0.00482568
7X10X10X3	100	7	0.005819	0.00582612	0.00432951	0.00532488
7X10X10X3	100	8	0.00707447	0.0044533	0.00507252	0.00553343
7X10X10X3	100	9	0.00486543	0.00355267	0.00255652	0.00365821
7X10X10X3	100	10	0.00334833	0.00532214	0.00318503	0.00395183
7X7X12X10X3	50	1	0.00326662	0.0060905	0.01121644	0.00685785

7X7X12X10X3	50	2	0.00369021	0.00544993	0.00680985	0.00531666
7X7X12X10X3	50	3	0.00224025	0.0046054	0.00528584	0.00404383
7X7X12X10X3	50	4	0.0039924	0.00462181	0.00778397	0.00546606
7X7X12X10X3	50	5	0.00882423	0.003775	0.01065873	0.00775266
7X7X12X10X3	50	6	0.00706277	0.00603356	0.00651681	0.00653771
7X7X12X10X3	50	7	0.0074165	0.00507499	0.00512346	0.00587165
7X7X12X10X3	50	8	0.00804515	0.00571454	0.00613803	0.00663257
7X7X12X10X3	50	9	0.00597039	0.00338853	0.00799073	0.00578322
7X7X12X10X3	50	10	0.00596918	0.00862581	0.01294343	0.00917947
7X7X12X10X3	100	1	0.00446389	0.00474255	0.01028014	0.00649553
7X7X12X10X3	100	2	0.00990021	0.00579328	0.01034467	0.00867939
7X7X12X10X3	100	3	0.00391922	0.0053353	0.00447797	0.0045775
7X7X12X10X3	100	4	0.00854676	0.00410318	0.00174964	0.00479986
7X7X12X10X3	100	5	0.00799575	0.00474512	0.00653342	0.00642476
7X7X12X10X3	100	6	0.00817445	0.00476004	0.00655652	0.006497
7X7X12X10X3	100	7	0.00396757	0.00347765	0.00293375	0.00345966
7X7X12X10X3	100	8	0.00368938	0.00439166	0.00378888	0.00395664
7X7X12X10X3	100	9	0.00388043	0.00444265	0.00872521	0.00568277
7X7X12X10X3	100	10	0.00873685	0.0067571	0.0066792	0.00739105
7X7X13X10X3	50	1	0.0026823	0.00429862	0.00507594	0.00401895
7X7X13X10X3	50	2	0.00423202	0.00653969	0.00617737	0.00564969
7X7X13X10X3	50	3	0.00292825	0.00683704	0.0042183	0.0046612
7X7X13X10X3	50	4	0.01552853	0.00638583	0.01352844	0.01181427
7X7X13X10X3	50	5	0.00295369	0.00679578	0.00395239	0.00456729
7X7X13X10X3	50	6	0.0038284	0.00568063	0.00685822	0.00545575
7X7X13X10X3	50	7	0.00870291	0.00547557	0.00768569	0.00728806
7X7X13X10X3	50	8	0.00559012	0.00407225	0.00598638	0.00521625
7X7X13X10X3	50	9	0.01448415	0.00749715	0.01212042	0.01136724
7X7X13X10X3	50	10	0.004652	0.00502709	0.01224502	0.00730804
7X7X13X10X3	100	1	0.00718362	0.00547046	0.01392207	0.00885872
7X7X13X10X3	100	2	0.00798069	0.00435553	0.00732896	0.00655506
7X7X13X10X3	100	3	0.00264784	0.00620627	0.00467338	0.00450916
7X7X13X10X3	100	4	0.00261	0.00628968	0.00465658	0.00451875
7X7X13X10X3	100	5	0.00779257	0.00542885	0.00593798	0.00638647
7X7X13X10X3	100	6	0.00780948	0.00542973	0.00594215	0.00639379
7X7X13X10X3	100	7	0.00948876	0.00472305	0.00877756	0.00766313
7X7X13X10X3	100	8	0.00872542	0.00337142	0.01047208	0.00752297
7X7X13X10X3	100	9	0.00847148	0.00730479	0.00940924	0.00839517
7X7X13X10X3	100	10	0.00870306	0.0053358	0.00265207	0.00556364
7X10X12X8X3	50	1	0.00486656	0.00704708	0.0027722	0.00489528
7X10X12X8X3	50	2	0.00859023	0.00589552	0.00291914	0.00580163
7X10X12X8X3	50	3	0.01278755	0.00424047	0.00688763	0.00797188
7X10X12X8X3	50	4	0.00894055	0.00403225	0.0109324	0.0079684
7X10X12X8X3	50	5	0.00367406	0.00330307	0.00217745	0.00305153
7X10X12X8X3	50	6	0.00692778	0.00597759	0.00243162	0.00511233

7X10X12X8X3	50	7	0.00610327	0.00377643	0.00204581	0.00397517
7X10X12X8X3	50	8	0.00318253	0.00529711	0.0033398	0.00393981
7X10X12X8X3	50	9	0.00673793	0.00687527	0.00509561	0.00623627
7X10X12X8X3	50	10	0.00530756	0.00525949	0.00509479	0.00522061
7X10X12X8X3	100	1	0.0039203	0.00529957	0.001897	0.00370562
7X10X12X8X3	100	2	0.00295498	0.00423887	0.0020021	0.00306531
7X10X12X8X3	100	3	0.00288506	0.00426379	0.00199532	0.00304806
7X10X12X8X3	100	4	0.00342262	0.00570615	0.00143719	0.00352199
7X10X12X8X3	100	5	0.00430291	0.00501022	0.00353842	0.00428385
7X10X12X8X3	100	6	0.00389535	0.00493761	0.00354277	0.00412524
7X10X12X8X3	100	7	0.00276324	0.0046416	0.00307674	0.00349386
7X10X12X8X3	100	8	0.00331097	0.00526007	0.00318209	0.00391771
7X10X12X8X3	100	9	0.00510393	0.00578628	0.00160881	0.00416634
7X10X12X8X3	100	10	0.00735798	0.0049047	0.00164528	0.00463598
7X11X13X10X3	50	1	0.00320484	0.00389862	0.00268562	0.00326303
7X11X13X10X3	50	2	0.0041116	0.00294217	0.0035624	0.00353872
7X11X13X10X3	50	3	0.00611343	0.00694774	0.01130121	0.00812079
7X11X13X10X3	50	4	0.00400062	0.00526217	0.01135127	0.00687135
7X11X13X10X3	50	5	0.00459113	0.00839954	0.00287689	0.00528918
7X11X13X10X3	50	6	0.00344124	0.00386136	0.00290603	0.00340288
7X11X13X10X3	50	7	0.00429587	0.00738551	0.00753592	0.00640577
7X11X13X10X3	50	8	0.00423605	0.00408026	0.00901809	0.00577813
7X11X13X10X3	50	9	0.0041935	0.00554362	0.00216966	0.00396893
7X11X13X10X3	50	10	0.0073184	0.00634624	0.0109167	0.00819378
7X11X13X10X3	100	1	0.00308586	0.00353781	0.00241856	0.00301408
7X11X13X10X3	100	2	0.00197354	0.00342222	0.00328286	0.00289287
7X11X13X10X3	100	3	0.0075139	0.0040386	0.00813009	0.00656086
7X11X13X10X3	100	4	0.00265783	0.004202	0.00398865	0.00361616
7X11X13X10X3	100	5	0.00577219	0.00264164	0.00792188	0.00544524
7X11X13X10X3	100	6	0.00599276	0.00268007	0.00809258	0.00558847
7X11X13X10X3	100	7	0.00367564	0.00570658	0.00466975	0.00468399
7X11X13X10X3	100	8	0.00698269	0.00263427	0.00517852	0.00493182
7X11X13X10X3	100	9	0.00349487	0.00242343	0.00302003	0.00297944
7X11X13X10X3	100	10	0.00371512	0.00409898	0.00582897	0.00454769

Tabla 12. Tamaños de red y ventana (I)

TIPOLOGÍA RED	*LOTE*	DESV. X	DESV. Y	DESV. Z	DESV. PROM.	TIEMPO ITERACIÓN	TIEMPO TOTAL
7X10X3	50	0.00451	0.00661	0.00461	0.00524	0.000538	19.379
7X10X3	50	0.00478	0.00683	0.00400	0.00520	0.000634	22.827
7X10X3	50	0.00472	0.00445	0.00700	0.00539	0.000616	22.177
7X10X3	50	0.00437	0.00842	0.00530	0.00603	0.000609	21.923
7X10X3	50	0.00404	0.00604	0.00491	0.00500	0.000619	22.266
7X10X3	50	0.00567	0.00496	0.00661	0.00575	0.000639	22.999
7X10X3	50	0.00527	0.00560	0.00181	0.00423	0.000596	21.436
7X10X3	50	0.00781	0.00505	0.00641	0.00643	0.000659	23.719
7X10X3	50	0.00533	0.00609	0.00481	0.00541	0.000623	22.428
7X10X3	50	0.00640	0.00414	0.00572	0.00542	0.000633	22.767
7X10X3	100	0.00434	0.00341	0.00362	0.00379	0.000671	24.154
7X10X3	100	0.00449	0.00475	0.00411	0.00445	0.000698	25.109
7X10X3	100	0.00574	0.00712	0.00549	0.00612	0.000695	24.998
7X10X3	100	0.00623	0.00620	0.00464	0.00569	0.000603	21.695
7X10X3	100	0.00279	0.00336	0.00496	0.00370	0.000648	23.339
7X10X3	100	0.00433	0.00650	0.00288	0.00457	0.000656	23.626
7X10X3	100	0.00434	0.00390	0.00379	0.00401	0.000670	24.103
7X10X3	100	0.00333	0.00280	0.00502	0.00372	0.000642	23.097
7X10X3	100	0.00415	0.00659	0.00321	0.00465	0.000667	24.006
7X10X3	100	0.00529	0.00432	0.00593	0.00518	0.000708	25.498
7X10X10X3	50	0.00440	0.00372	0.00342	0.00385	0.000658	23.671
7X10X10X3	50	0.00525	0.00378	0.00224	0.00376	0.000660	23.734
7X10X10X3	50	0.00358	0.00416	0.00496	0.00423	0.000695	25.002
7X10X10X3	50	0.00367	0.00358	0.00380	0.00368	0.000649	23.317
7X10X10X3	50	0.00319	0.00313	0.00625	0.00419	0.000655	23.558
7X10X10X3	50	0.00652	0.00419	0.00362	0.00478	0.000666	23.951
7X10X10X3	50	0.00353	0.00453	0.00715	0.00507	0.000691	24.858
7X10X10X3	50	0.00488	0.00475	0.00409	0.00457	0.000705	25.348
7X10X10X3	50	0.00385	0.00382	0.00182	0.00316	0.000701	25.216
7X10X10X3	50	0.00385	0.00382	0.00182	0.00316	0.000738	26.523
7X10X10X3	100	0.00336	0.00225	0.00312	0.00291	0.000768	27.563
7X10X10X3	100	0.00289	0.00173	0.00307	0.00256	0.000705	25.326
7X10X10X3	100	0.00335	0.00286	0.00148	0.00257	0.000733	26.299
7X10X10X3	100	0.00449	0.00282	0.00325	0.00352	0.000716	25.716
7X10X10X3	100	0.00291	0.00216	0.00187	0.00232	0.000758	27.229
7X10X10X3	100	0.00291	0.00276	0.00402	0.00323	0.000734	26.343
7X10X10X3	100	0.00406	0.00435	0.00315	0.00385	0.000765	27.478
7X10X10X3	100	0.00471	0.00290	0.00337	0.00366	0.000745	26.739
7X10X10X3	100	0.00374	0.00239	0.00195	0.00269	0.000756	27.145
7X10X10X3	100	0.00286	0.00345	0.00237	0.00289	0.000775	27.819
7X7X12X10X3	50	0.00237	0.00368	0.00871	0.00492	0.000724	26.038
7X7X12X10X3	50	0.00264	0.00402	0.00614	0.00426	0.000753	27.054
7X7X12X10X3	50	0.00199	0.00356	0.00379	0.00312	0.000739	26.573

7X7X12X10X3	50	0.00296	0.00313	0.00602	0.00404	0.000757	27.201
7X7X12X10X3	50	0.00600	0.00293	0.00777	0.00557	0.000773	27.801
7X7X12X10X3	50	0.00503	0.00299	0.00482	0.00428	0.000796	28.615
7X7X12X10X3	50	0.00410	0.00401	0.00466	0.00425	0.000739	26.576
7X7X12X10X3	50	0.00465	0.00371	0.00465	0.00434	0.000763	27.438
7X7X12X10X3	50	0.00511	0.00271	0.00590	0.00457	0.000755	27.135
7X7X12X10X3	50	0.00560	0.00631	0.00984	0.00725	0.000769	27.631
7X7X12X10X3	100	0.00341	0.00271	0.00683	0.00432	0.000845	30.350
7X7X12X10X3	100	0.00699	0.00327	0.00734	0.00587	0.000824	29.574
7X7X12X10X3	100	0.00296	0.00301	0.00394	0.00330	0.000828	29.722
7X7X12X10X3	100	0.00465	0.00307	0.00157	0.00310	0.000858	30.809
7X7X12X10X3	100	0.00539	0.00331	0.00476	0.00449	0.000866	31.089
7X7X12X10X3	100	0.00547	0.00330	0.00476	0.00451	0.000877	31.469
7X7X12X10X3	100	0.00269	0.00231	0.00187	0.00229	0.000827	29.672
7X7X12X10X3	100	0.00332	0.00324	0.00365	0.00340	0.000786	28.214
7X7X12X10X3	100	0.00296	0.00334	0.00577	0.00402	0.000815	29.273
7X7X12X10X3	100	0.00538	0.00378	0.00626	0.00514	0.000798	28.634
7X7X13X10X3	50	0.00195	0.00318	0.00284	0.00265	0.000765	27.496
7X7X13X10X3	50	0.00329	0.00505	0.00437	0.00424	0.000761	27.373
7X7X13X10X3	50	0.00240	0.00428	0.00265	0.00311	0.000794	28.526
7X7X13X10X3	50	0.01121	0.00402	0.00819	0.00781	0.000792	28.479
7X7X13X10X3	50	0.00283	0.00464	0.00318	0.00355	0.000803	28.852
7X7X13X10X3	50	0.00319	0.00426	0.00499	0.00414	0.000817	29.385
7X7X13X10X3	50	0.00787	0.00400	0.00478	0.00555	0.000825	29.669
7X7X13X10X3	50	0.00337	0.00331	0.00391	0.00353	0.000863	31.027
7X7X13X10X3	50	0.00848	0.00443	0.00800	0.00697	0.000861	30.946
7X7X13X10X3	50	0.00355	0.00330	0.00874	0.00520	0.000873	31.390
7X7X13X10X3	100	0.00578	0.00335	0.00926	0.00613	0.000793	28.457
7X7X13X10X3	100	0.00676	0.00347	0.00576	0.00533	0.000791	28.386
7X7X13X10X3	100	0.00209	0.00352	0.00327	0.00296	0.000798	28.633
7X7X13X10X3	100	0.00205	0.00353	0.00328	0.00296	0.000847	30.423
7X7X13X10X3	100	0.00578	0.00359	0.00449	0.00462	0.000842	30.244
7X7X13X10X3	100	0.00579	0.00359	0.00449	0.00463	0.000768	27.577
7X7X13X10X3	100	0.00916	0.00420	0.00665	0.00667	0.000793	28.451
7X7X13X10X3	100	0.00557	0.00267	0.00641	0.00488	0.000795	28.538
7X7X13X10X3	100	0.00623	0.00346	0.00690	0.00553	0.000868	31.147
7X7X13X10X3	100	0.00536	0.00387	0.00199	0.00374	0.000849	30.494
7X10X12X8X3	50	0.00316	0.00467	0.00177	0.00320	0.000731	26.261
7X10X12X8X3	50	0.00559	0.00383	0.00228	0.00390	0.000761	27.342
7X10X12X8X3	50	0.00548	0.00278	0.00525	0.00450	0.000754	27.089
7X10X12X8X3	50	0.00519	0.00297	0.00595	0.00470	0.000754	27.099
7X10X12X8X3	50	0.00300	0.00251	0.00178	0.00243	0.000770	27.679
7X10X12X8X3	50	0.00455	0.00421	0.00178	0.00351	0.000776	27.897
7X10X12X8X3	50	0.00436	0.00308	0.00163	0.00302	0.000795	28.588
7X10X12X8X3	50	0.00283	0.00342	0.00210	0.00278	0.000805	28.934

7X10X12X8X3	50	0.00402	0.00546	0.00321	0.00423	0.000817	29.380
7X10X12X8X3	50	0.00402	0.00390	0.00389	0.00394	0.000828	29.753
7X10X12X8X3	100	0.00285	0.00373	0.00166	0.00274	0.000879	31.559
7X10X12X8X3	100	0.00207	0.00327	0.00194	0.00243	0.000882	31.663
7X10X12X8X3	100	0.00207	0.00349	0.00192	0.00249	0.000901	32.360
7X10X12X8X3	100	0.00261	0.00380	0.00145	0.00262	0.000917	32.925
7X10X12X8X3	100	0.00299	0.00368	0.00258	0.00309	0.000942	33.822
7X10X12X8X3	100	0.00287	0.00361	0.00259	0.00302	0.000876	31.459
7X10X12X8X3	100	0.00249	0.00330	0.00293	0.00290	0.000760	27.300
7X10X12X8X3	100	0.00240	0.00293	0.00204	0.00246	0.000788	28.296
7X10X12X8X3	100	0.00407	0.00350	0.00121	0.00293	0.000790	28.348
7X10X12X8X3	100	0.00406	0.00309	0.00125	0.00280	0.000814	29.211
7X11X13X10X3	50	0.00219	0.00240	0.00204	0.00221	0.000802	28.828
7X11X13X10X3	50	0.00508	0.00248	0.00313	0.00357	0.000822	29.533
7X11X13X10X3	50	0.00463	0.00362	0.00891	0.00572	0.000738	26.530
7X11X13X10X3	50	0.00277	0.00280	0.00746	0.00434	0.000758	27.233
7X11X13X10X3	50	0.00345	0.00484	0.00225	0.00351	0.000762	27.392
7X11X13X10X3	50	0.00301	0.00279	0.00194	0.00258	0.000765	27.514
7X11X13X10X3	50	0.00439	0.00370	0.00602	0.00470	0.000790	28.394
7X11X13X10X3	50	0.00369	0.00250	0.00749	0.00456	0.000804	28.895
7X11X13X10X3	50	0.00390	0.00381	0.00174	0.00315	0.000813	29.231
7X11X13X10X3	50	0.00568	0.00346	0.00927	0.00614	0.000817	29.373
7X11X13X10X3	100	0.00236	0.00230	0.00204	0.00223	0.000786	28.216
7X11X13X10X3	100	0.00183	0.00262	0.00260	0.00235	0.000799	28.671
7X11X13X10X3	100	0.00593	0.00295	0.00698	0.00528	0.000849	30.477
7X11X13X10X3	100	0.00223	0.00339	0.00328	0.00296	0.000827	29.685
7X11X13X10X3	100	0.00436	0.00205	0.00562	0.00401	0.000863	30.970
7X11X13X10X3	100	0.00452	0.00194	0.00585	0.00410	0.000895	32.133
7X11X13X10X3	100	0.00267	0.00320	0.00327	0.00305	0.000836	29.999
7X11X13X10X3	100	0.00472	0.00195	0.00358	0.00342	0.000808	28.992
7X11X13X10X3	100	0.00239	0.00197	0.00223	0.00219	0.000806	28.923
7X11X13X10X3	100	0.00335	0.00268	0.00462	0.00355	0.000818	29.357

Tabla 13.Tamaños de red y ventana (II)

REPETICIONES	PRUEBA	ERROR X	ERROR Y	ERROR Z	ERROR PROMEDIO
2	1	0.00386709	0.00334561	0.00707517	0.00476262
2	2	0.00392526	0.00345608	0.00719225	0.00485786
2	3	0.0026719	0.00364683	0.00338767	0.00323546
2	4	0.00348282	0.00463408	0.00319745	0.00377145
2	5	0.00497905	0.00418181	0.00610324	0.00508804
2	6	0.00469983	0.00245927	0.00328871	0.0034826
2	7	0.00384762	0.00604221	0.00293844	0.00427609
2	8	0.00369223	0.00382038	0.00226907	0.00326056
2	9	0.00556937	0.00384204	0.00288634	0.00409925
2	10	0.0052906	0.00389773	0.0022227	0.00380368
5	1	0.00441854	0.00344559	0.00272001	0.00352805
5	2	0.00494172	0.00326435	0.0024576	0.00355456
5	3	0.00439376	0.00583413	0.00517921	0.0051357
5	4	0.00384464	0.00468286	0.00376124	0.00409625
5	5	0.00613403	0.0080333	0.00423379	0.00613371
5	6	0.00433283	0.00261019	0.00557562	0.00417288
5	7	0.00439747	0.00386573	0.00362914	0.00396411
5	8	0.00383456	0.00236191	0.00280129	0.00299925
5	9	0.00704026	0.0032715	0.00654305	0.00561827
5	10	0.00573102	0.00532329	0.00570412	0.00558614
10	1	0.00539588	0.00565114	0.00391421	0.00498708
10	2	0.00373765	0.00301186	0.00201797	0.00292249
10	3	0.00425194	0.00458025	0.00272197	0.00385139
10	4	0.0049457	0.00441832	0.00263533	0.00399978
10	5	0.00879249	0.00309283	0.01096496	0.00761676
10	6	0.00498471	0.00348078	0.00471599	0.00439383
10	7	0.0049709	0.0041896	0.00802333	0.00572794
10	8	0.00535644	0.00351469	0.00331577	0.0040623
10	9	0.00415664	0.0068029	0.00577596	0.0055785
10	10	0.00703405	0.00357011	0.00594601	0.00551672

Tabla 14. Pasos en el entrenamiento (I).

REP.	PRUEBA	DESV. X	DESV. Y	DESV. Z	DESV. PROMEDIO	TIEMPO ITERACIÓN	TIEMPO TOTAL
2	1	0.00284	0.00255	0.00490	0.00343	0.000669	24.030
2	2	0.00285	0.00262	0.00500	0.00349	0.000735	26.376
2	3	0.00191	0.00243	0.00252	0.00229	0.000759	27.237
2	4	0.00254	0.00364	0.00298	0.00306	0.000699	25.094
2	5	0.00321	0.00301	0.00443	0.00355	0.000701	25.147
2	6	0.00330	0.00173	0.00272	0.00258	0.000856	30.718
2	7	0.00304	0.00405	0.00259	0.00323	0.000747	26.816
2	8	0.00325	0.00288	0.00206	0.00273	0.000727	26.091
2	9	0.00420	0.00238	0.00277	0.00312	0.000746	26.772
2	10	0.00382	0.00253	0.00166	0.00267	0.000753	27.022
5	1	0.00370	0.00285	0.00183	0.00279	0.000675	24.297
5	2	0.00386	0.00238	0.00194	0.00273	0.000689	24.812
5	3	0.00343	0.00518	0.00295	0.00385	0.000694	24.983
5	4	0.00292	0.00278	0.00304	0.00291	0.000718	25.845
5	5	0.00480	0.00439	0.00314	0.00411	0.000736	26.493
5	6	0.00353	0.00213	0.00428	0.00331	0.000657	23.652
5	7	0.00357	0.00293	0.00263	0.00304	0.000668	24.059
5	8	0.00271	0.00177	0.00244	0.00230	0.000682	24.542
5	9	0.00374	0.00222	0.00450	0.00349	0.000693	24.959
5	10	0.00357	0.00327	0.00381	0.00355	0.000733	26.377
10	1	0.00370	0.00416	0.00239	0.00341	0.000659	23.702
10	2	0.00234	0.00230	0.00193	0.00219	0.000684	24.604
10	3	0.00280	0.00281	0.00229	0.00263	0.000672	24.168
10	4	0.00352	0.00350	0.00189	0.00297	0.000764	27.489
10	5	0.00484	0.00258	0.00997	0.00579	0.000700	25.194
10	6	0.00321	0.00278	0.00298	0.00299	0.000710	25.543
10	7	0.00355	0.00351	0.00625	0.00444	0.000734	26.407
10	8	0.00306	0.00232	0.00225	0.00254	0.000774	27.855
10	9	0.00294	0.00384	0.00367	0.00348	0.000734	26.433
10	10	0.00335	0.00299	0.00332	0.00322	0.000753	27.098

Tabla 15. Pasos en el entrenamiento (II).

MINI-BATCH	PRUEBA	ERROR X	ERROR Y	ERROR Z	ERROR PROMEDIO
1	1	0.0039202	0.00607884	0.00410304	0.00470069
1	2	0.00395414	0.0110641	0.00556786	0.00686203
1	3	0.00449567	0.00826668	0.00547192	0.00607809
1	4	0.00853021	0.00766007	0.00416436	0.00678488
1	5	0.00787844	0.00647771	0.00376279	0.00603965
10	1	0.00371801	0.00337741	0.00340936	0.00350159
10	2	0.00349231	0.00508479	0.0052345	0.00460387
10	3	0.00487832	0.00982015	0.00245209	0.00571685
10	4	0.00410081	0.00230368	0.00293	0.0031115
10	5	0.00467153	0.0038268	0.00808288	0.00552707
10	6	0.00527119	0.00317354	0.00511328	0.00451934
10	7	0.00269397	0.00517179	0.00353462	0.00380013
10	8	0.00401706	0.00299946	0.00285464	0.00329039
10	9	0.00635719	0.00392012	0.00268683	0.00432138
10	10	0.00392284	0.00573121	0.0017463	0.00380012
50	1	0.00392105	0.00478857	0.00359621	0.00410194
50	2	0.00451797	0.00508897	0.0037931	0.00446668
50	3	0.00466488	0.00387785	0.00680732	0.00511668
50	4	0.00548809	0.00413904	0.00263007	0.00408573
50	5	0.00363208	0.00531631	0.00293705	0.00396181
50	6	0.00505222	0.00482096	0.00519016	0.00502112
50	7	0.00298754	0.00566379	0.00239237	0.00368123
50	8	0.00300782	0.00565824	0.00338815	0.00401807
50	9	0.0063383	0.00456939	0.00358761	0.00483177
50	10	0.00544491	0.00391226	0.00198797	0.00378171

Tabla 16. Nueva gestión de memoria (I).

MINI-BATCH	PRUEBA	DESV. X	DESV. Y	DESV. Z	DESV. PROM	TIEMPO ITERACION	TIEMPO TOTAL
1	1	0.0031	0.0042	0.0029	0.0034	0.1600	2872.004
1	2	0.0036	0.0112	0.0034	0.0061	0.1625	2916.648
1	3	0.0032	0.0046	0.0043	0.0040	0.1648	2958.859
1	4	0.0056	0.0056	0.0029	0.0047	0.1670	2998.366
1	5	0.0057	0.0042	0.0025	0.0041	0.1695	3041.849
10	1	0.0029	0.0021	0.0021	0.0024	0.0215	386.538
10	2	0.0034	0.0028	0.0020	0.0028	0.0216	388.166
10	3	0.0051	0.0043	0.0022	0.0039	0.0206	370.464
10	4	0.0033	0.0016	0.0020	0.0023	0.0209	375.446
10	5	0.0030	0.0025	0.0045	0.0033	0.0224	401.676
10	6	0.0038	0.0024	0.0038	0.0033	0.0218	390.506
10	7	0.0024	0.0032	0.0025	0.0027	0.0220	394.463
10	8	0.0038	0.0022	0.0020	0.0026	0.0244	437.873
10	9	0.0036	0.0029	0.0018	0.0027	0.0233	418.407
10	10	0.0040	0.0032	0.0017	0.0030	0.0224	401.687
50	1	0.0030	0.0034	0.0024	0.0029	0.0040	72.519
50	2	0.0029	0.0033	0.0026	0.0029	0.0044	78.145
50	3	0.0028	0.0026	0.0051	0.0035	0.0043	76.882
50	4	0.0040	0.0028	0.0021	0.0030	0.0044	78.916
50	5	0.0023	0.0034	0.0018	0.0025	0.0042	76.265
50	6	0.0028	0.0029	0.0037	0.0031	0.0043	77.752
50	7	0.0025	0.0038	0.0020	0.0028	0.0044	78.802
50	8	0.0019	0.0034	0.0020	0.0024	0.0042	75.395
50	9	0.0048	0.0030	0.0025	0.0034	0.0043	77.500
50	10	0.0035	0.0026	0.0017	0.0026	0.0042	76.267

Tabla 17. Nueva gestión de memoria (II).

11 REFERENCIAS

[1] J. Weng and Y. Zhang, “Developmental Robots - A New Paradigm,” 2002.

[2] A. Prieto, A. Romero, F. Bellas, R. Salgado and R. J. Duro. “Introducing Separable Utility Regions in a Motivational Engine for Cognitive Developmental Robotics,” Ferrol, A Coruna, Spain, 2018.

[3] R. Sutton and A. Barto, *Reinforcment Learning*. 2018.

[4] J. B. Hamrick, “Analogues of mental simulation and imagination in deep learning,” *Curr. Opin. Behav. Sci.*, vol. 29, pp. 8–16, 2019.

[5] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*. 2015.

[6] José Antonio Becerra, Alejandro Romero, Francisco Bellas and Richard J. Duro. “Motivational Engine and Long Term Memory Coupling within a Cognitive Architecture for Life-long Open-ended Learning,” Ferrol, A Coruna, Spain, 2019.

[7] F. Bellas, R. J. Duro, A. Faiña, and D. Souto, “Multilevel darwinist brain (MDB): Artificial evolution in a cognitive architecture for real robots,” 2010.

[8] S. Ruder, “An overview of gradient descent optimization algorithms,” 2016.

[9] D. P. Kingma and J. Lei Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.”

[10] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.”

[11] M. Abadi *et al.*, “TensorFlow: A system for large-scale machine learning.” 2016.

[12] W. H. Calvin, “Neural Darwinism. The Theory of Neuronal Group Selection. Gerald M. Edelman. Basic Books, New York, 1987.,” *Science*, vol. 240, no. 4860, p. 1802, Jun. 1988.

[13] M. Conrad, “Evolutionary learning circuits,” *J. Theor. Biol.*, 1974.

[14] J.-P. Changeux, P. Courrege, and A. Danchin, “A Theory of the Epigenesis of Neuronal Networks by Selective Stabilization of Synapses,” *Proc. Natl. Acad. Sci.*, 1973.